NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

ROBOTIC MANIPULATOR CALIBRATION: DEVELOPMENT OF A GENERAL METHOD FOR KINEMATIC MODEL PARAMETER SET IDENTIFICATION

by

Robert J. Burger

December, 1994

Thesis Advisor: Morris Driels

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1994	3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE ROBOTIC MANIPULATOR CALIBRATION: DEVELOPMENT OF A GENERAL METHOD FOR KINEMATIC MODEL PARAMETER SET IDENTIFICATION	5. FUNDING NUMBERS	
6. AUTHOR(S) Robert J. Burger		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER	

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*
A general method for establishing a kinematic model of a robotic manipulator with either a full or partial pose calibration system is developed. The theory applicable to modeling of mechanisms is introduced, as is robotic manipulator calibration. Given a general over-specified kinematic model, a method is developed, with the associated algorithms for a six degree of freedom manipulator, that identifies non-unique parameter sets for any given partial pose measurement system. This method is applied and demonstrated on three existing calibration methods.

14. SUBJECT TERMS Robot Manipulator, Kinematic Model, Robotic Manipulator Calibration			15. NUMBER OF PAGES 114
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

298-102

ROBOTIC MANIPULATOR CALIBRATION: DEVELOPMENT OF A GENERAL METHOD FOR KINEMATIC MODEL PARAMETER SET IDENTIFICATION

Robert J. Burger
Lieutenant, United States Navy
B. S., United States Naval Academy, 1986

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

and

MECHANICAL ENGINEER

from the

NAVAL POSTGRADUATE SCHOOL
December 1994

Author: [illegible]
Robert J. Burger

Approved by: [illegible]
Morris Driels, Thesis Advisor

[illegible]
Matthew D. Kelleher, Chairman
Department of Mechanical Engineering

ABSTRACT

A general method for establishing a kinematic model of a robotic manipulator with either a full or partial pose calibration system is developed. The theory applicable to modeling of mechanisms is introduced, as is robotic manipulator calibration. Given a general over-specified kinematic model, a method is developed, with the associated algorithms for a six degree of freedom manipulator, that identifies non-unique parameter sets for any given partial pose measurement system. This method is applied and demonstrated on three existing calibration methods.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

The study of robotic manipulator calibration is a continuing quest to improve the accuracy of the manipulator, thereby increasing its effectiveness and usefulness in the workplace. Manipulator accuracy is the precision to which the robotic system can place the end effector, or tool, in a commanded position and orientation, defined as the pose, within the working volume [Ref. 1]. A commanded pose is not one that has been previously taught to the manipulator, but one that is directed from an outside source, such as a computer data base. In order to achieve a commanded pose, the robotic system must calculate the joint variables required for the given pose location and orientation within the work space. This is accomplished by solving the inverse solution to the kinematic model of the manipulator. Accuracy of general industrial robots can be as poor as 10mm between the commanded and actual poses. [Ref. 2].

Repeatability is defined as the precision to which a manipulator can reacquire a previously taught pose. This differs from accuracy in that the manipulator is placed in the desired pose within the working volume, and the associated joint variables are then recorded for later use during the work cycle. In contrast to the aforementioned accuracy statistic, current manipulators have a high degree of repeatability, on the order of 0.1mm. [Ref. 1]

As industry demands increase for both computerized process management and computerized simulation and reconfiguration of both repetitive tasks and unique, single occurrence tasks, robotic manipulators are required to be programmed off-line and to share control programs. Off-line programming and shared control programs require a much higher degree of accuracy than currently exists -- a degree of accuracy more in line with current repeatability statistics.

Factors that impact the accuracy of a robotic manipulator include: (1) inaccurate knowledge of the robot geometry and kinematic parameters, (2) link and joint compliance, (3) steady state servo controller errors, (4) gear backlash, and (5) temperature variations [Ref. 3]. Since it would be extremely expensive to manufacture a robot with the required stiffness and tolerances to negate the above issues and achieve the desired accuracy, the resulting errors must be identified and accounted for. The calibration process is the identification and correction of these errors. For the purposes of this work, calibration will only deal with the accurate identification of a manipulators actual geometry, or kinematic parameters.

There are four basic steps to the calibration process: [Ref. 4,5]

1. A closed chain kinematic model of the manipulator and measurement system is developed. This involves the determination of the identifiable kinematic parameters that form the complete model. This complete model is used to determine an error quantity based on the nominal kinematic parameter set and the actual measured set. The nominal parameter values are provided by the manipulator manufacturer and the measurement system specifications and location.

2. Experimental measurements of the robot pose (partial or complete) are taken. These measurements are a function of the actual parameter set.

3. The actual kinematic parameter values are identified by systematically adjusting the nominal parameters until the model predictions match the measured values, driving the error functions, defined in the modeling phase to zero.

4. The final step is to incorporate the identified parameter set into the controlling software for the manipulator.

This work addresses the first step of the calibration process, the development of the closed chain kinematic model. Standardized and reliable approaches exist for modeling any given manipulator itself. Yet, in order to perform a calibration, the model employed must also incorporate the kinematics of the measurement system (full or partial) that is used to get the experimental data. Determining this complete closed chain model is a process that is neither obvious nor straightforward. The most difficult issue in closed chain model development is whether or not an existing kinematic parameter can be identified. The result of this thesis is a standardized method for determining the complete closed-chain kinematic model for any manipulator and measurement system. This was accomplished by defining a general, over-specified model, then conducting a calibration simulation with this model in order to generate the system Jacobian. Finally, parameters were systematically removed until a full rank Jacobian resulted, thus defining the complete kinematic model.

II. THEORY

A. KINEMATIC MODELING

In order to understand how a manipulator achieves a commanded pose, one must understand the format of the kinematic model. When a commanded pose is given to the manipulator, it is specified relative to a world coordinate frame in the workspace. This world coordinate frame is then moved through each link of the manipulator by a coordinate transformation operation until it reaches the tool. The result is a description of the commanded pose in joint space. This set of coordinate transformations makes up the kinematic model of the manipulator. This section deals with the development of the coordinate transformations and the kinematic model. The theory discussed and some of the diagrams used in this chapter closely follow discussions by Paul [Ref. 9] and Mooring, Roth and Driels [Ref. 10].

1. General Coordinate Transformations

As just described, the fundamental basis for kinematic modeling of a manipulator is the successive movement of a set of coordinate axes from one position and orientation in space to another. There are two building blocks that comprise the general coordinate transformation, the rotation transformation and the translation transformation. For the rotation transformation, consider two coordinate frames, where frame 2 has been rotated an angle ϕ about the z axis of frame 1 (Figure 1). Note that the frames remain coincident and share the same z axis, allowing a two dimensional representation of the x-y plane (Figure 2). As illustrated in Figure 2, a point p is described in frame 2 by the vector

$$\vec{p}_2 = p_{x2}\vec{i}_2 + p_{y2}\vec{j} + p_{z2}\vec{k} \tag{1}$$

where $\vec{i}_2$, $\vec{j}_2$ and $\vec{k}_2$ represent the x, y and z unit direction vectors of coordinate frame 2. Likewise, point p can be described in frame 1 by the vector

$$\vec{p}_1 = p_{x1}\vec{i}_1 + p_{y1}\vec{j}_1 + p_{z1}\vec{k}_1 \tag{2}$$

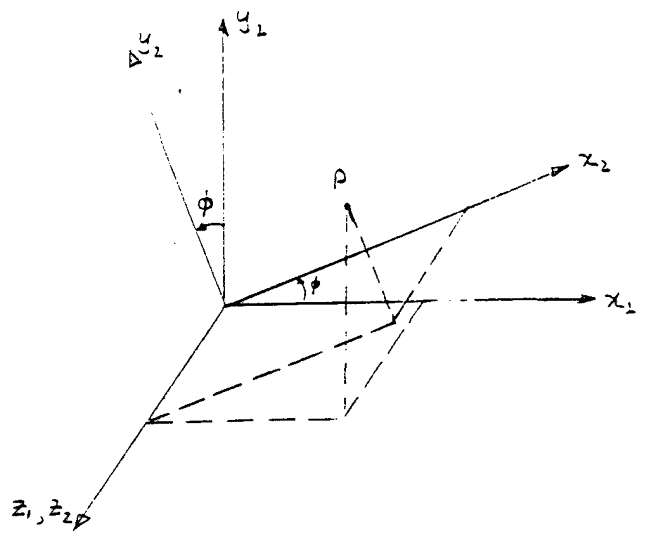

Figure 1. Frame rotation about the z axis.

The position of point p in frame 1 can then be related to the position of point p in frame 2 by:

$$p_{x1} = p_{x2}\cos\phi - p_{y2}\sin\phi \tag{3}$$
$$p_{y1} = p_{x2}\sin\phi + p_{y2}\cos\phi \tag{4}$$
$$p_{z1} = p_{z2} \tag{5}$$

Transferring these equations into matrix form results in equation 6:

$$\begin{bmatrix} p_{x1} \\ p_{y1} \\ p_{z1} \end{bmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_{x2} \\ p_{y2} \\ p_{z2} \end{bmatrix} \tag{6}$$

or

$$\vec{p}_1 = \text{rot}(z,\phi)\vec{p}_2 \tag{7}$$

The notation rot(z, ϕ) represents a rotation about the z axis of angle ϕ. Similar analyses can be conducted for rotations about the x and y axes. The results of these analyses will be presented later.

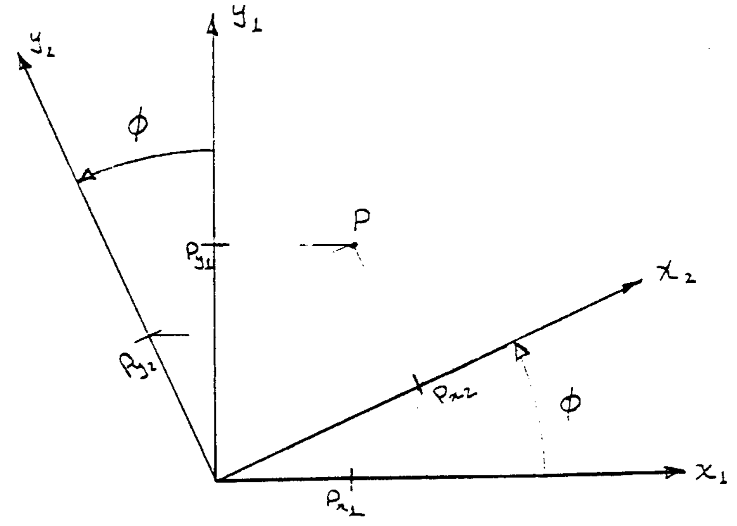

Figure 2. 2-D view of frame rotated @ the *z* axis

The translation transformation can be thought of as moving the origin of a coordinate frame to a new point described by the vector $\vec{v}$, as illustrated in Figure 3, where

$$\vec{v} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{8}$$

If the previous rotation and translation were to occur, the equation would be of the form

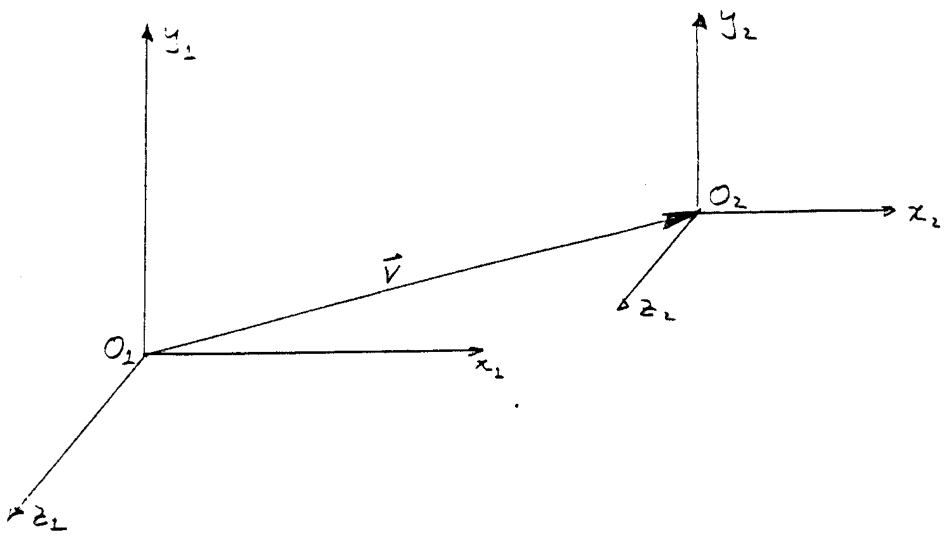

Figure 3. Translation of a coordinate frame

$$\vec{p}_1 = \mathrm{rot}(z,\phi)\vec{p}_2 + \vec{v} \qquad (9)$$

For convenience, equation 9 is expressed as a single matrix equation:

$$\begin{bmatrix} p_{x1} \\ p_{y1} \\ p_{z1} \\ w \end{bmatrix} = T \begin{bmatrix} p_{x2} \\ p_{y2} \\ p_{z2} \\ w \end{bmatrix} \qquad (10)$$

Where T is a 4x4 augmented matrix of the rotation and translation transforms and *w* is a non-zero scale factor. For the example represented in equation 9, T is of the form:

$$T = \begin{bmatrix} \cos\phi & -\sin\phi & 0 & x \\ \sin\phi & \cos\phi & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & w \end{bmatrix} \qquad (11)$$

The matrix T is defined as the transformation matrix, and in the above example is a product of the rotation about the *z* axis and translation transformation matrices. Equations 12 through 15 list the general form of the translation matrix and all the rotation transformation matrices. Unless dealing with a perspective transformation, the scale factor *w* equals one.

$$\mathrm{Trans}(x,y,z) = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & w \end{bmatrix} \tag{12}$$

$$\mathrm{Rot}(x,\psi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi & 0 \\ 0 & \sin\psi & \cos\psi & 0 \\ 0 & 0 & 0 & w \end{bmatrix} \tag{13}$$

$$\mathrm{Rot}(y,\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & w \end{bmatrix} \tag{14}$$

$$\mathrm{Rot}(z,\phi) = \begin{bmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & w \end{bmatrix} \tag{15}$$

As evidenced in the previous example, a general coordinate transformation can be the product of individual rotation and translation transformations. The transformation represented in equation 16 can be thought of as a series of individual transformations, that when read left to right, represent a transformation operation on, or relative to, the resulting coordinate frame of the previous transformations.

$$\mathrm{Trans(x,y,z)Rot(y,90)Rot(z,90)} = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & w \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & w \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & w \end{bmatrix} \tag{16}$$

Therefore, equation 16 represents a transformation that is a translation of the base frame origin in the *x,y,z* directions respectively, then a 90 degree rotation about that new frame's *y* axis, then a 90 degree rotation about the *z* axis of the most recent coordinate frame.

2. Euler Transformations

A six parameter transformation is required to transform from one coordinate frame that is fixed in space to another frame, also fixed in space. That is, three rotations are required to align the respective axes, and three translations are required to bring the origins coincident. As may have been realized in the previous section, the order that the transformations occur is important. This begs for the use of a standard frame to frame transformation to avoid confusion.

The standard used in this work is the Euler transformation (Equation. 17), also known as the Roll, Pitch, Yaw transformation and will be represented as RPYT(ϕ,θ,ψ,x,y,z) [Ref. 11].

$$\mathrm{RPYT}(\phi,\theta,\psi,x,y,z) = \mathrm{Rot}(z,\phi)\mathrm{Rot}(y,\theta)\mathrm{Rot}(x,\psi)\mathrm{Trans}(x,y,z) \tag{17}$$

This can be read as a rotation ϕ about the *z* axis, followed by a rotation θ about the new *y* axis, followed by a rotation ψ about the new *x* axis, then an *x, y, z* translation in the direction of the new *x, y, z* axes.

3. Denavit-Hartenburg Transformations

A manipulator is made up of a sequence of links that are connected to each other by joints. A coordinate frame is associated with each link and a transformation matrix is required to describe the relative positions of successive frames. Since the links fix the geometric relationship between joints, there can be a reduction in the number of rotations and translations required by the Euler transformation. The Denavit-Hartenburg transformation provides an accepted systematic approach used to define link geometry and coordinate frame placement, resulting in reduced parameter transformations between link coordinate frames. For an *n* degree of freedom manipulator, there will be *n* links and *n* joints. Figure 4 illustrates a PUMA 560 six degree of freedom manipulator. Link and joint labeling starts at the base of the manipulator and progresses through to the tool, with joint *n* preceding link *n*. The base of the manipulator is defined as link 0, and not considered to be one of the *n* links. Therefore link 1 is joined to link 0 by joint 1 and link 2 is joined to link 1 by joint 2, and so on.

A generic link geometry can be characterized by two dimensions, the link length and link twist angle, illustrated in Figure 5. The link length, a_n, is the length of the common normal between the joint axes *n* and $n+1$. The link twist angle, α_n, lies in a plane that is perpendicular to the common normal and located at the intersection of the common normal and joint axis $n+1$. This plane, therefore, contains joint axis $n+1$ and a line that is a parallel projection of joint axis *n*. The twist angle, α_n, is the angle formed by the intersection of these two lines in the plane.

The relationship of one link to the next, connected by a joint, is described by the distanced, d_n, and angle, θ_n, between links, as illustrated in Figure 6. Each joint axis *n* will have two common normal lines intersecting it, one for link *n-1* and one for link *n*. The distance between the intersection points of the two common normal lines on the joint axis is the distance between links, d_n. The angle between links, θ_n, lies in a plane perpendicular to joint axis *n* at the intersection of joint axis *n* and the common normal for link *n-1* (N_1). Similar to the definition for α_n, this plane contains the line N_1 and a parallel projection of the common normal for link *n* (N_2 and N_2'). The angle between links is then the counterclockwise angle between N_1 and N_2'.

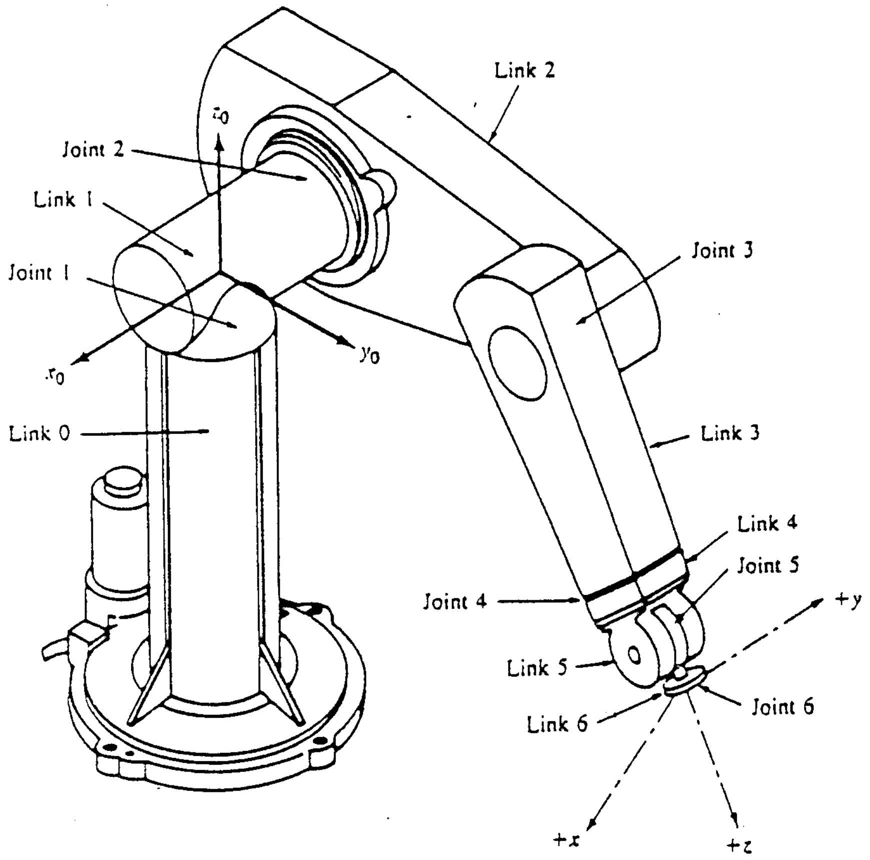

Figure 4. PUMA 560 robot manipulator with links and joints labeled

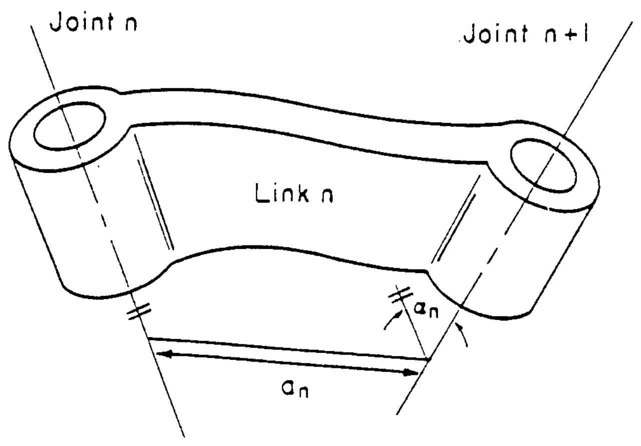

Figure 5. Generic manipulator link

With the link and joint geometries defined, coordinate frames can be assigned to each link. For a revolute joint, illustrated in Figure 6, the angle between links, θ_n, is the joint variable. The origin of the coordinate frame for link *n* is located at the intersection of joint axis $n+1$ and the common normal for link *n* (N_2 in Figure 6). The x_n axis is aligned along the common normal (N_2), and the z_n axis is placed along joint axis $n+1$. The joint variable θ_n is in the zero position when x_n and x_{n-1} are parallel. There are two exceptions to this method of placement, specifically when two successive joint axes either intersect or are parallel. For intersecting axes (*n* and $n+1$), the link length a_n is set to zero. The frame origin is placed at the intersection point of the joint axes. The x_n axis direction is normal to the plane that contains both joint axes *n* and $n+1$. For parallel axes there is not a unique common normal, and the distance between joints, d_n, can be arbitrarily chosen. Therefore the frame origin is placed on the $n+1$ joint axis at a point that will make the distance between joints zero for the next link, d_{n+1}, whose coordinate frame origin is defined.

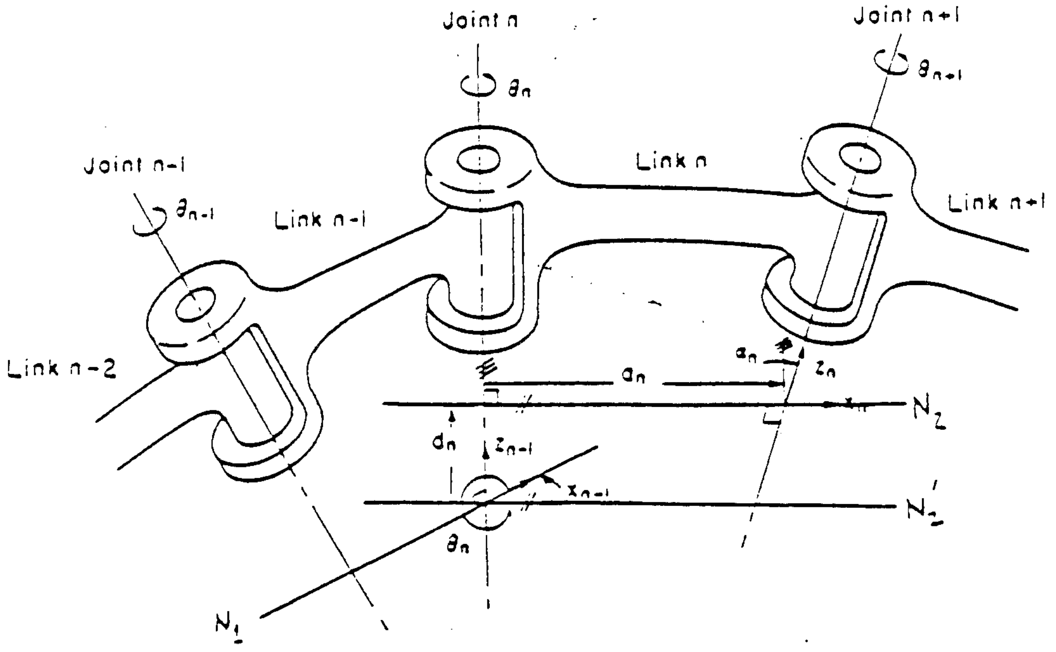

Figure 6. Revolute joint with link parameters

For a prismatic joint, illustrated in Figure 7, the joint variable is the joint distance d_n. Unlike a revolute joint, where the circular path of a point rotating about an axis defines both the plane in which the rotation occurs and the location in space of the axis about which the point rotates, the path of a point moving along a straight line , as in a prismatic joint, only defines an axis direction and not the location of the axis in space. Therefore, the link length, a_n, is meaningless for a prismatic joint. As a result, the coordinate frame for a prismatic joint is placed coincident with the next defined link coordinate frame origin. The z_n axis direction lies along the $n+1$ joint axis with the x_n direction perpendicular to both the $n+1$ joint axis and the prismatic joint direction.

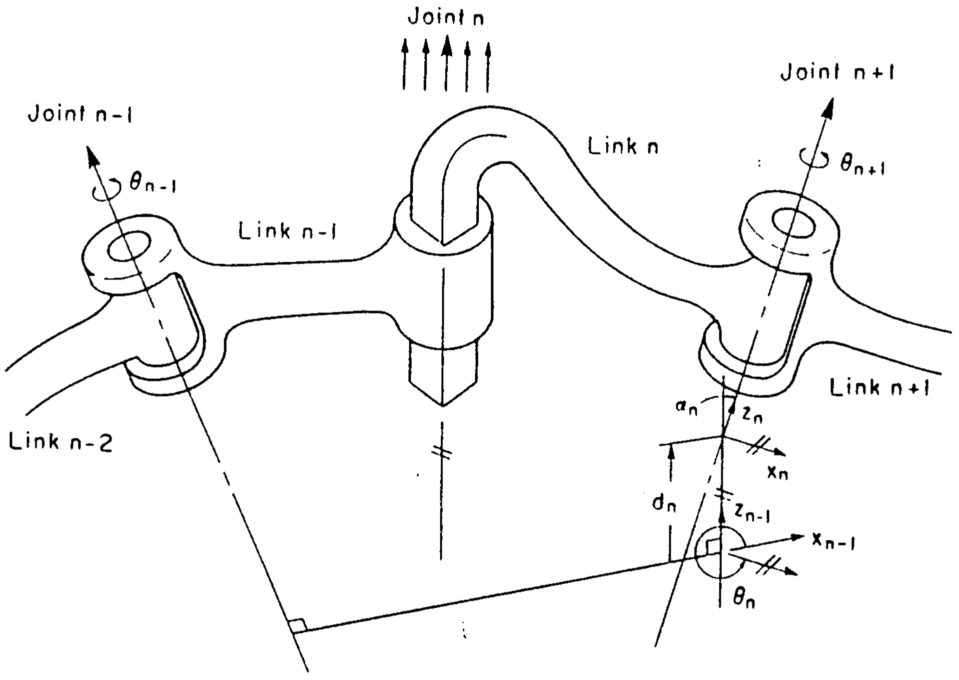

Figure 7. Prismatic joint with link parameters

Having defined the geometric relationships between links and the joint coordinate frame, the transformation matrices can be developed. The Denavit-Hartenburg transformation uses the translation and rotation matrices discussed previously to step from frame to frame in a logical manner. Referring to Figure 6, the transformation can be completed in four steps:

1. Rotate an angle θ_n about the z_{n-1} axis.
2. Translate a distance d_n along the z_{n-1} axis.
3. Translate a distance a_n along the new x axis (same direction as x_n).

4. Rotate an angle α_n about the x_n axis, aligning the z axis with the $n+1$ joint axis.

The resulting transformation matrix is described by equation 18:

$$A_n = Rot(z,\theta_n)Trans(z,d_n)Trans(x,a_n)Rot(x,\alpha_n) \tag{18}$$

4. Modified Denavit-Hartenburg Transformation

The standard Denavit-Hartenburg transform provides an accurate model of a manipulator for most forward kinematic solutions. But in the case of manipulator calibration, where the kinematic parameters become the variables, potential problems arise for parallel or nearly parallel consecutive axes.

Consider two parallel axes where a common normal does not exist between the joint axes, and the coordinate frame placement is a matter of convenience. If in fact the two axes are not actually parallel, then a common normal does exist between the two, therefore fixing the position of the coordinate frame. This could result in a disproportionately large change in the kinematic parameters that then cascades through the remaining links and joints. These disproportional changes cause numerical instability in the parameter identification process.

The Modified Denavit-Hartenburg transformation solves this problem by removing the requirement for a common normal between joint axes. As illustrated in Figure 8, this is accomplished by defining a plane that is perpendicular to joint axis *n* and intersects joint axis *n* at the origin of the *n-1* coordinate frame. The intersection of joint axis *n* with this plane defines the origin of coordinate frame *n*, and a line in the plane from frame *n-1* to frame *n*. [Ref. 12]

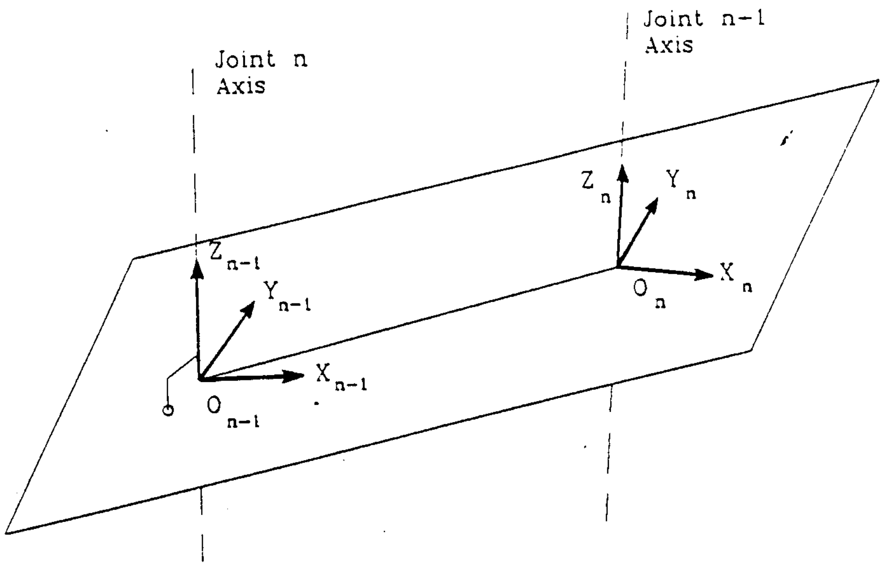

Figure 8. Modified Denavit-Hartenburg transformation, after Ref. [12]

The transformation equation is then

$$A_n = \mathrm{Rot}(z,\theta_n)\mathrm{Trans}(x,r_n)\mathrm{Rot}(x,\alpha_n)\mathrm{Rot}(y,\beta_n) \qquad (19)$$

which consists of:

1. A rotation of angle θ_n about the z_{n-1} axis

2. A translation of distance r_n along the line connecting the coordinate frame origins *n-1* and *n*, which is also the new *x* axis direction.

3. A rotation of α_n about the new *x* axis

4. A rotation of β_n about the new *y* axis to align the z_n axis direction with joint axis *n+1*.

Equation 20 can then be used for any type of manipulator coordinate frame transformation where β_n is set to zero for standard Denavit-Hartenburg transformations applied to joints with non-parallel consecutive axes, and d_n is set to zero for the transformations between joints with nominally parallel axes.

$$A_n = Rot(z,\theta_n)Trans(z,d_n)Trans(x,a_n)Rot(x,\alpha_n)Rot(y,\beta_n) \tag{20}$$

Equation 20 is the Modified Denavit-Hartenburg transformation equation that is used in all calibration procedures in this work.

5. The Kinematic Chain

Once homogeneous transformation matrices(whether Euler or Modified Denavit-Hartenburg) have been developed for a single coordinate frame transformation, it is time to describe the pose of the end effector of a manipulator with respect to the world coordinate frame of the work space. As mentioned previously, this is accomplished by stepping from frame to frame, beginning at the world coordinate frame and ending at the end effector coordinate frame. This path of homogeneous transformations is commonly referred to as the kinematic chain. The kinematic chain is formed by sequentially post-multiplying the transformation matrix chain by the next link homogeneous transform, as in equation 21.

$$T_W{}^E = A_W{}^0 A_0{}^1 \ldots A_{n-1}{}^n A_n{}^E \tag{21}$$

The 'A' matrices are either Euler transforms, as for $A_W{}^0$ and $A_n{}^E$, or Modified Denavit-Hartenburg for $A_0{}^1 \ldots A_{n-1}{}^n$. The notation used in equation (21) represents a transformation from the world coordinate frame, *W* (subscript), to the end effector, *E* (superscript). The kinematic chain is illustrated graphically in Figure 9.

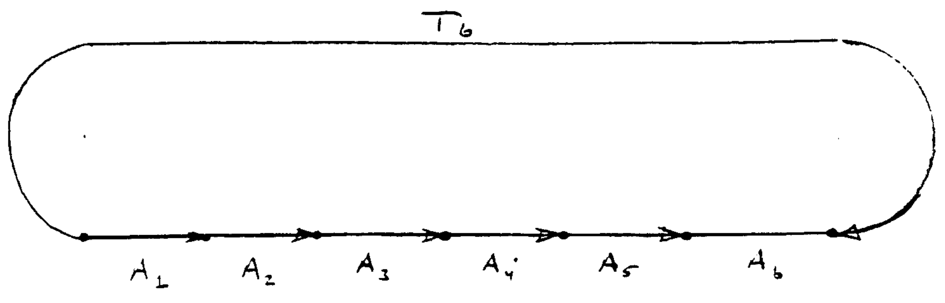

Figure 9. Kinematic chain

6. General 30 Parameter PUMA Manipulator Model

Armed with the appropriate tools to model a manipulator, a model can be generated for the PUMA 560 six degree of freedom manipulator illustrated in Figure 4. The PUMA is used as the reference manipulator throughout this work. Figure 10 shows the PUMA with the Modified Denavit-Hartenburg coordinate frames attached, as well as the world coordinate frame, F_W, the manufacturers designated base frame, F_b, and an end effector frame, F_6. The transformations from frame (1) to frame (5) each contain the standard four parameters from the Modified Denavit-Hartenburg transformation, but the transforms from the world frame to frame (1) via the base frame need to be considered carefully since potential parameter dependencies exist.

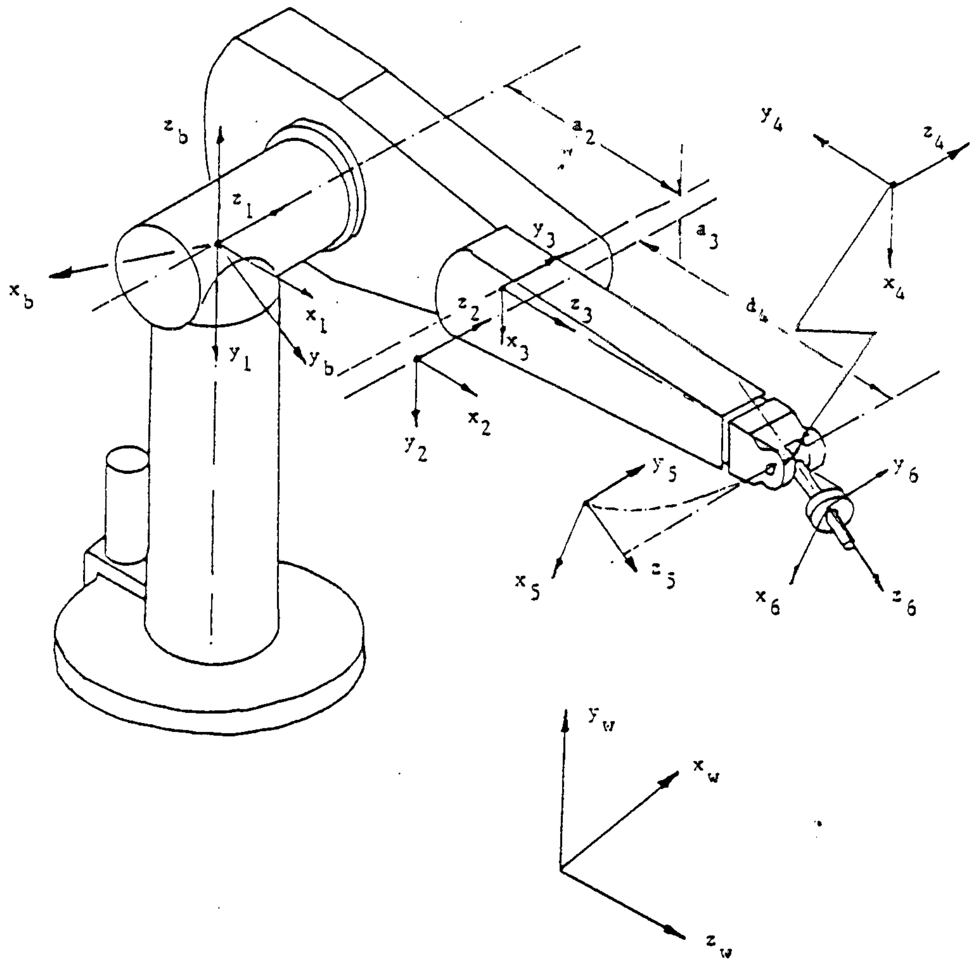

Figure 10. PUMA 560 coordinate frame allocation

In order to ensure the minimum number of parameters required to move from the world coordinate frame to frame (1), two possible paths may be taken. [Ref. 13] The first path involves three Denavit-Hartenburg transformations, shown in Figure 11:

1. A four parameter transformation from the world coordinate frame to an intermediate frame, designated frame (0).

$$A_0 = \mathrm{Rot}(z,\theta_0)\mathrm{Trans}(z,d_0)\mathrm{Trans}(x,a_0)\mathrm{Rot}(x,\alpha_0) \qquad (22)$$

2. A Denavit-Hartenburg transform from frame (0) to frame (b) that only requires two of the four Denavit-Hartenburg parameters, θ_b, d_b.

$$A_b = \mathrm{Rot}(z,\theta_b)\mathrm{Trans}(z,d_b) \qquad (23)$$

3. A four parameter Denavit-Hartenburg transformation from frame (b) to frame (1)

$$A_1 = \mathrm{Rot}(z,\theta_1)\mathrm{Trans}(z,d_1)\mathrm{Trans}(x,a_1)\mathrm{Rot}(x,\alpha_1) \qquad (24)$$

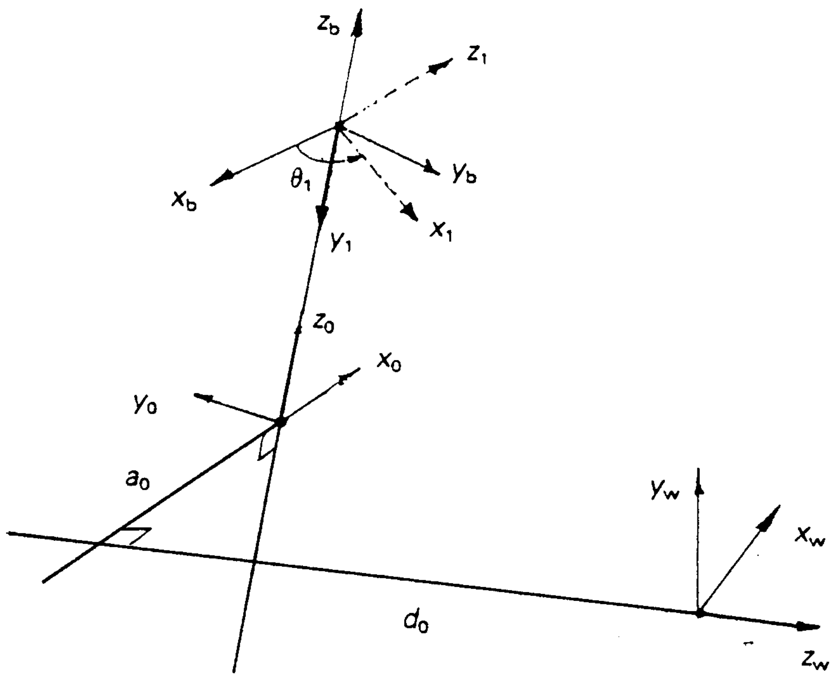

Figure 11. Base transformations

The result of this path is an eight parameter transform from the world coordinate frame to frame (1), since θ_1 and ϕ_1 are both about the z_0 axis and cannot be identified independently, and d_1 and d_b are both along the z_0 axis and cannot be distinguished independently either.

The second path involves only two transforms to get from the world coordinate frame to frame (1).

1. A full six parameter Euler transform from the world frame to the base frame.

$$A_b = \mathrm{Rot}(z,\phi_b)\mathrm{Rot}(y,\theta_b)\mathrm{Rot}(x,\psi_b)\mathrm{Trans}(x_b,y_b,z_b) \quad (25)$$

2. A Denavit-Hartenburg transform from the base frame to frame 1.

$$A_1 = \mathrm{Rot}(z,\theta_1)\mathrm{Trans}(z,d_1)\mathrm{Trans}(x,a_1)\mathrm{Rot}(x,\alpha_1) \quad (26)$$

As with the first path, a total of eight parameters are required to get from the world frame to frame (1), since q_1 can be resolved into f_b, q_b, y_b and d_1 can be resolved into x_b, y_b, z_b. Note that both paths require eight parameters, but not the same set of eight. For this work the second path is used.

The last transform in the kinematic chain is from frame (5) to the tool, frame (6). This is a full frame to frame transformation and requires a six parameter Euler transformation.

$$A^6 = \mathrm{Rot}(z,\phi_6)\mathrm{Rot}(y,\theta_6)\mathrm{Rot}(x,\psi_6)\mathrm{Trans}(x_6,y_6,z_6) \quad (27)$$

Table 1 provides the nominal values of the 30 parameters required for the PUMA 560, where the parmeters in bold type are not identifiable.

Link #	$\Delta\theta$	d	a	α	β
0	180	-397	-383	90	**0**
1	0	467	0	-90	**0**
2	0	**0**	431.9	0	0
3	0	149.1	-20.3	90	**0**
4	0	433	0	-90	**0**
5	0	0	0	90	**0**
ϕ_6	θ_6	ψ_6	p_x	p_y	p_z
0	0	0	0	0	135

Table 1. Nominal kinematic parameters for PUMA 560

7. 28 Parameter 5R1P Manipulator

Another manipulator model used in this work is the Driels and Pathre 5R1P manipulator shown in Figure 12. The 5R1P is a six degree of freedom manipulator, similar to the PUMA 560, but with 5 revolute joints and one prismatic joint. As discussed in Section 3, the length a_3 for the prismatic joint 3 has no meaning and is defined to be zero. With the location of the prismatic joint coordinate frame (frame 3) determined by the common normal between axes 4 and 5, and placed coincident to the origin of frame (4), the length d_4 is also, by definition, equal to zero. The kinematic model for the 5R1P is then a 28 parameter model with the nominal parameters listed in Table 2.

Link #	$\Delta\theta$	d	a	α	β
0	180	-397	-383	90	0
1	0	467	0	-90	0
2	0	250	0	90	0
3	0	0	0	30	0
4	0	0	0	-90	0
5	0	0	0	90	0
ϕ_6	θ_6	ψ_6	p_x	p_y	p_z
90	0	0	0	0	134

Table 2. Nominal kinematic parameters for 5R1P manipulator

B. PARAMETER IDENTIFICATION METHODOLOGY

1. Experimental Measurement

Assuming an appropriate, complete kinematic model of the manipulator has been developed with nominal kinematic parameter values, actual parameter value identification or calibration can be performed. Experimental measurements of the end effector pose are taken. These can be either full or partial pose measurements, depending upon the measurement system used for calibration. Partial pose measurements do not measure all six pose elements (three rotations and three translations). Complete kinematic models for partial pose measurement systems have fewer than the full 30 parameter PUMA model, depending on the constraints applied by the measurement system. For each pose, the measured pose data and the associated joint angles are recorded. This process is repeated for additional poses until an adequate data base is collected. An adequate data base is dependent on the number of kinematic parameters to be identified (unknowns), and the number of known pieces of data measured on each pose. For a full pose measurement system, a minimum of N/6 different pose measurements must be taken, where N is the number of parameters to be identified. This number must then be increased to account for measurement noise and the degree of accuracy desired.

2. Identification

Once experimental measurements have been taken, with joint angles recorded for each pose, the kinematic parameter values can be identified. One approach to identifying the kinematic parameters is to determine the differential relationship between the pose variables, P, and the kinematic parameters, K [Refs. 2,15]. The differential pose variables are related to the differential kinematic parameters, or parameter error vector, by the Identification Jacobian.

$$\delta P = J\delta K \tag{22}$$

The differential pose variable vector, δP, is a *p x 1* column vector, where p is equal to 6 for a full pose measurement system:

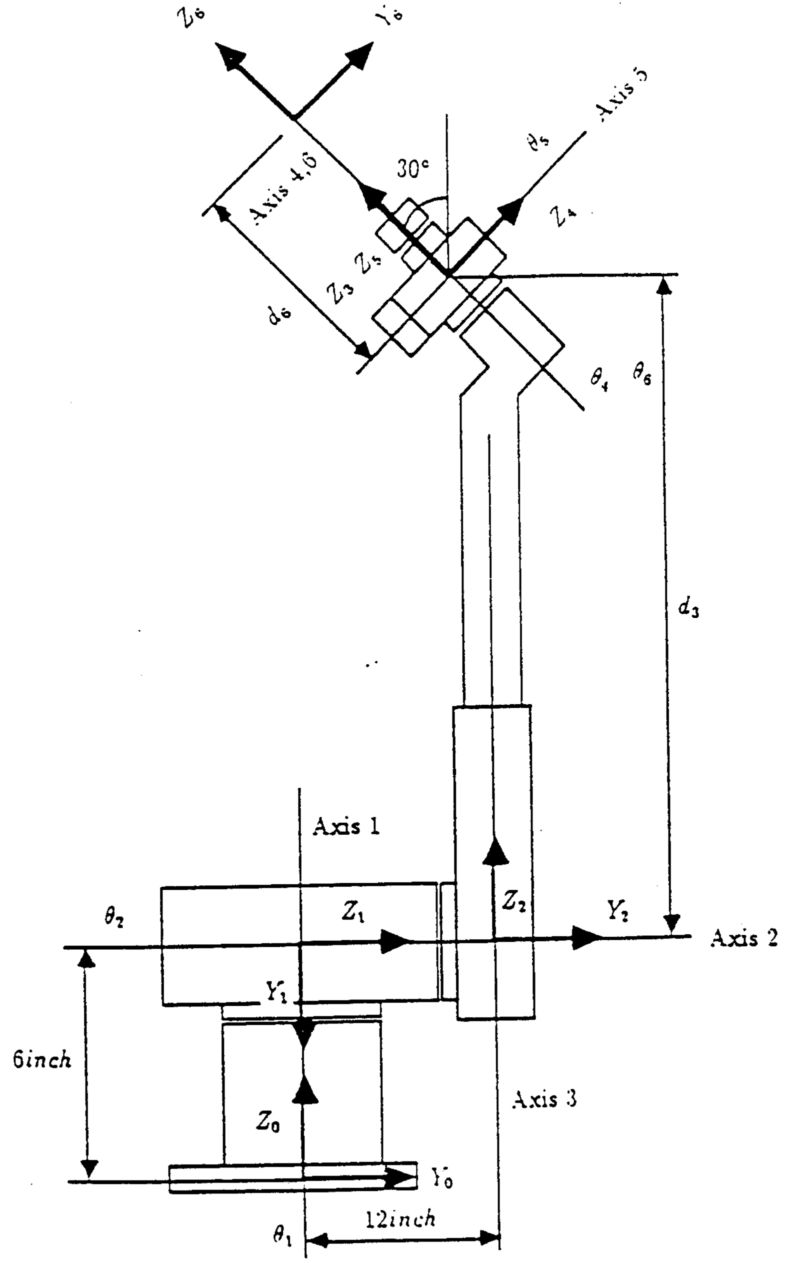

Figure 12. 5R1P Robot manipulator & coordinate frames, from Ref. [14].

$$\delta P = \begin{bmatrix} d\delta_x \\ d\delta_y \\ d\delta_z \\ dp_x \\ dp_y \\ dp_z \end{bmatrix} \tag{22}$$

The Jacobian is a $p \ x \ n$ matrix, where n is the number of parameters in the complete kinematic model. The parameter error vector, δK, is an $n \ x \ 1$ column vector of the differential parameters. For multiple pose measurements each differential relationship is appended on to the previous one, resulting in the equation:

$$\delta T = J\delta K \tag{23}$$

Here δT is a $(p \ x \ m)$ x 1 vector, J is a $(p \ x \ m)$ x n matrix, δK is still an n x 1 vector, and m is the number of pose measurements taken. Equation 23 can then be inverted using the Jacobian pseudo-inverse (since the Jacobian is not a square matrix), to solve for the kinematic parameters directly:

$$\delta K = [J^T J]^{-1} J^T \delta T \tag{24}$$

To avoid solving equation 23 directly, a multidimensional optimization routine is used to minimize a user defined error function based on the difference between the pose data calculated from the nominal kinematic parameters and the actual pose data.

3. Optimization

An accurate approximation of actual kinematic parameters can be achieved by systematically changing the nominal parameter set in a non-linear, least squares sense. This is done until a minimum value is reached for the error function relating the actual and nominal kinematic parameters [Ref. 16]. This non-linear least squares minimization of a pose data error function has been employed with success on numerous full and partial pose robot calibration systems [Refs. 6,7,8,13].

The IMSL library routine ZXSSQ is a FORTRAN based algorithm that will minimize a function over a given variable set using a Levenburg-Marquardt non-linear least squares method. At each iteration in ZXSSQ, the next estimate of the variable set is based on a numerical approximation of the Jacobian. The Jacobian approximation is arrived at using forward or central difference methods around the current parameter set. Upon satisfying the designated convergence exit criteria, ZXSSQ returns the most recent estimate of the parameter set. Figure 13 illustrates the program flow of ZXSSQ where N is the size of the variable set vector, x. ZXSSQ uses an external user-defined subroutine that contains the error function to be minimized. Using this subroutine, it calculates the finite difference gradient. Based on that gradient, it then re-evaluates the function and tests for convergence.

The error function supplied to ZXSSQ is generated by calculating the differential transformation matrix, Δ, as described by Paul [Ref. 7]:

$$\Delta = \begin{bmatrix} 0 & -\delta_z & \delta_y & d_x \\ \delta_z & 0 & -\delta_x & d_y \\ -\delta_y & \delta_x & 0 & d_z \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{25}$$

The Δ matrix is generated by calculating the forward kinematic solution of the manipulator, T^E, using the nominal kinematic parameters, T^{Ec}, for each set of joint angles. The forward kinematic solution is again calculated, but using the experimental pose measurement data, T^{Em}. The difference between T^{Ec} and T^{Em} is the difference matrix ΔT.

The elements of the ΔT matrix, t_{ij}, are used to calculate the elements of the differential transformation matrix, Δ, equations 26-31, where the indices i and j represent the respective row and column of the matrix. The Δ matrix is only valid when the T^{Ec} and T^{Em} matrices are almost equal.

$$\delta_x = \left|\frac{t_{32}-t_{23}}{2}\right| \tag{26}$$

$$\delta_y = \left|\frac{t_{13}-t_{31}}{2}\right| \tag{27}$$

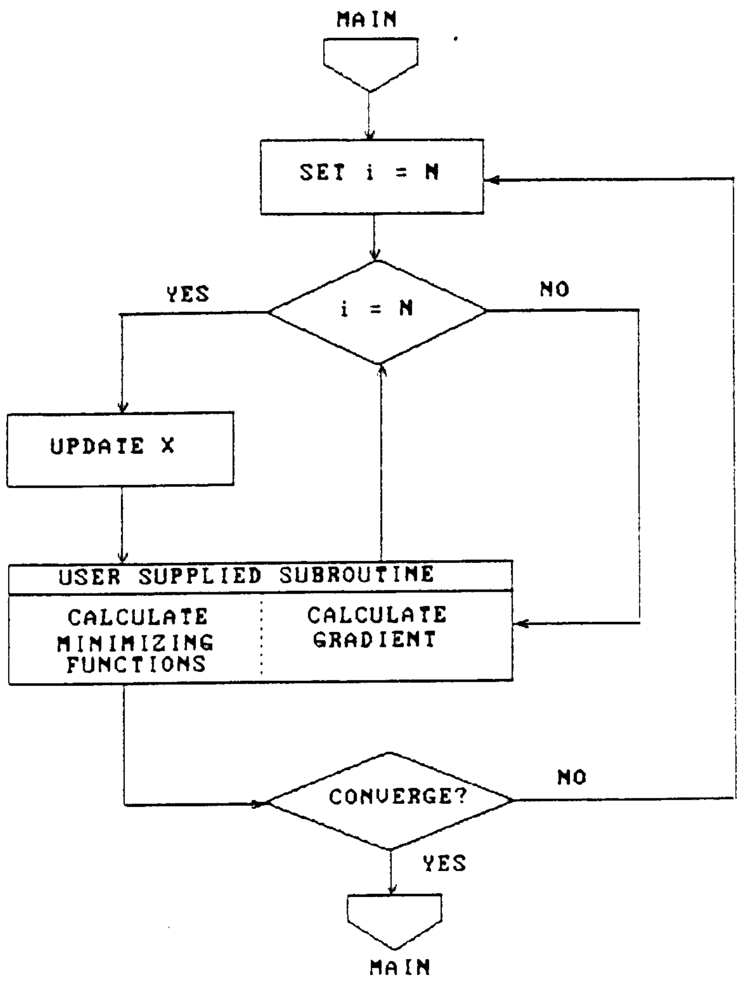

Figure 13. Program flow for ZXSSQ

$$\delta_z = \left|\frac{t_{12}-t_{21}}{2}\right| \qquad (28)$$

$$d_x = t_{14} \qquad (29)$$

$$d_y = t_{24} \qquad (30)$$

$$d_z = t_{34} \qquad (31)$$

The error function is the sum of the squares of the elements of the Δ matrices for the entire set of measured poses, equations 32 and 33.

$$\sum_{i=1}^{m}\sum_{j=1}^{6}(f_{ij}(x))^2 \qquad (32)$$

$$\begin{aligned} f_{i,1}(x) &= \delta_x \\ f_{i,2}(x) &= \delta_y \\ f_{i,3}(x) &= \delta_z \\ f_{i,4}(x) &= d_x \\ f_{i,5}(x) &= d_y \\ f_{i,6}(x) &= d_z \end{aligned} \qquad (33)$$

Note that for a partial pose measurement the error function is a sum of the squares of only those elements in the Δ matrix that correspond to the actual measurement data taken. In other words, if only position data is measured, and no orientation data, then the error function will only use d_x, d_y, and d_z for all the pose measurements.

In order to ensure that possition and orientation are equally weighted and represented when minimizing the error function, all elements in Δ need to be the same order of magnitude. Since the orientation error elements, δ_x, δ_y, and δ_z, are not the same order of magnitude as the possition elements, they are multiplied by a scale factor to match and ensure a proportional optimization.

4. General Identification Algorithms

As stated in the previous section, ZXSSQ uses an externally defined error function. The program ID6 contains the error function calculation in the subroutine PUMA_ARM. ID6 program flow is illustrated in Figure 14. ID6 reads an input file with nominal kinematic parameter values for the particular model being calibrated. These values are then used to initialize the variable vector x in ZXSSQ. ID6 also reads the measurement data with the associated joint variable data, from another file, and passes all this information in the proper format to ZXSSQ. The final parameter values from ZXSSQ are output to a file with residual error values of position and orientation.

A simulation of the calibration process is conducted to ensure the identification routine is correct, to predict the number of experimental pose measurements required to complete the identification, and to estimate the resulting accuracy of the manipulator after calibration. The sequence of simulation algorithms used is illustrated in Figure 15. Program JOINT is used to generate random joint data within the constraints defined by the work space and the pose measurement system. The program then writes this data to PUMA_VAR. Program POSE calculates the forward kinematic pose matrix solution for the associated joint data sets from PUMA_VAR, and writes the combined data to PUMA_POS. ID6 has been described previously. VERIFY confirms the accuracy of the identification routine by calculating a total position and orientation error from a comparison of the forward kinematic solution, using the nominal parameter values and the identified values.

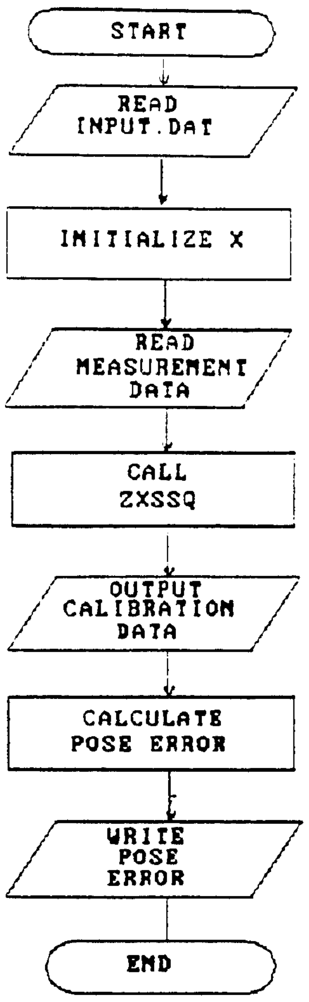

Figure 14. Program flow for ID6, form Ref. [8].

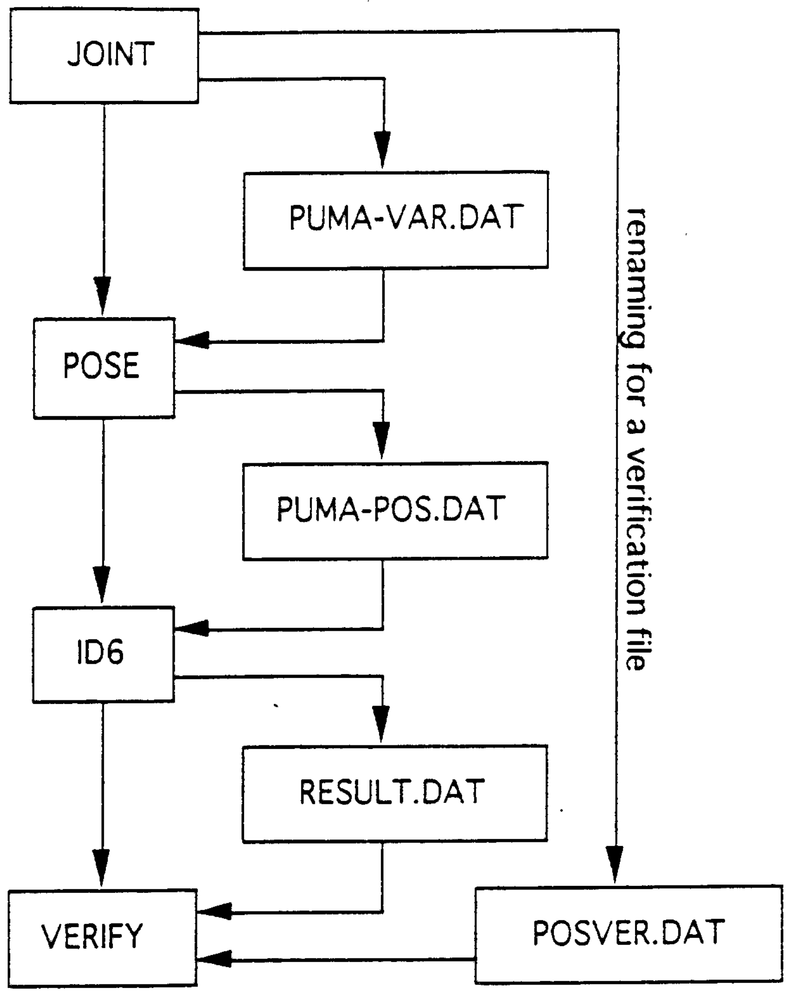

Figure 15. Simulation process flowchart, from Ref. [7].

III. COMPLETE MODEL DETERMINATION

A. INTRODUCTION

As previously stated, determining the complete closed-chain kinematic model of a robot manipulator and calibration measurement system is the most difficult portion of the calibration process. A complete model, as defined for this work, is the kinematic model of the system with the minimum number of parameters required to achieve convergence to the actual parameter values during identification. If there are not enough parameters in the model, it does not fully describe the manipulator. Yet, with too many parameters the model is over-specified, and the identification routine will not converge to an accurate solution. Therefore, all the parameters in the model must be identifiable for it to be complete. This also implies that an over-specified model contains parameters that cannot be identified independently. Arriving at the number and type of identifiable parameters in a kinematic model is where a majority of the work and thought reside in model development.

Although well established, closed-chain models exist for full pose measurement systems, less capable, partial pose measurement systems result in a reduction in the number of identifiable parameters from the full pose model. Determining the constraints and dependencies that exist in any given system is a difficult task since no clear, standardized approach exists to determine the set of parameters required to define a complete model. Therefore, model development frequently becomes an iterative, trial and error process.

B. THE CLOSED-CHAIN MODEL

As stated previously, the closed-chain model incorporates both the manipulator and measurement system kinematics, Figure 16. Here the manipulator kinematics have been well defined using Denavit-Hartenburg or Modified Denavit-Hartenburg methods. The measurement system kinematics uniquely define the position and orientation of the end effector relative to the world coordinate frame. From this point on, the world coordinate frame may be referred to as the measurement system frame (F_M), since the world frame is usually placed at some convenient location in the measurement system.

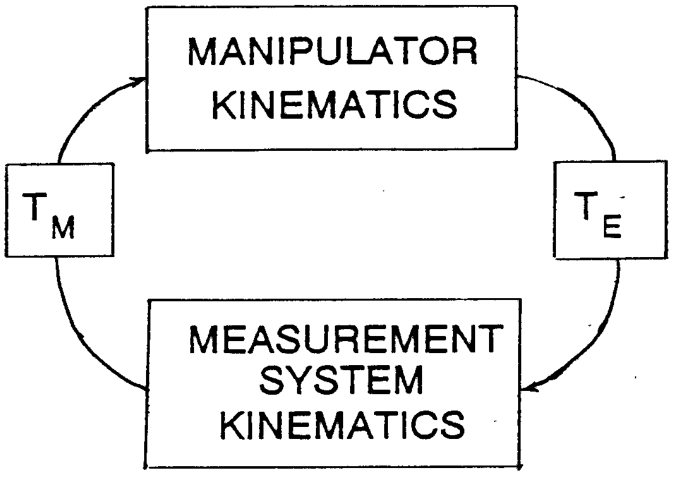

Figure 16. The closed chain model, after Ref. [8].

An in depth analysis of the transformation from the independently located world or measurement system coordinate frame to the manipulator base frame (T_M in Figure 16) was completed in Chapter II. Recall that dependencies existed between the world-to-base transformation parameters and the manipulator kinematic parameters. Two different paths were taken, resulting in different parameter sets but the same number of parameters.

The end effector transformation, T_E, is a six parameter transformation from the last link of the manipulator (*n-1* for an *n* degree of freedom robot) to the independent location of the tool frame. The Euler transformations between the manipulator and measurement system are illustrated in Figure 17.

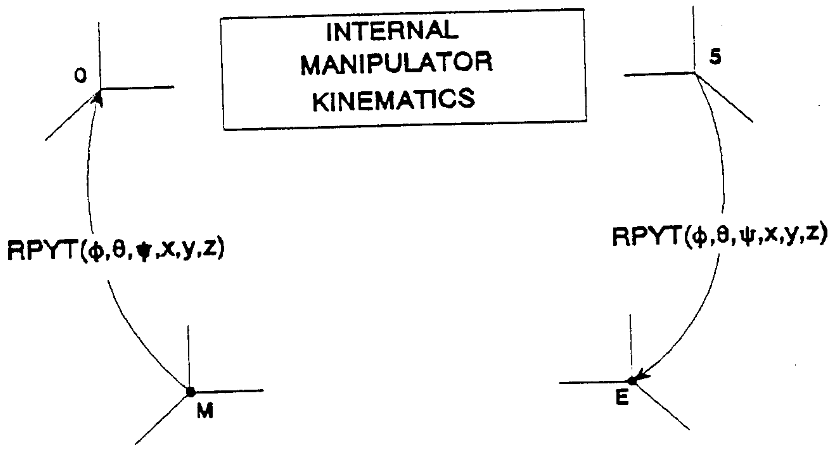

Figure 17. Manipulator and measurement system transformations, from Ref. [8].

Parameter dependencies may exist between the end effector transform and the manipulator kinematics. Careful end effector coordinate definition and placement, however, can usually avoid these dependencies. Additional parameter dependencies in frames T_M and T_E may exist due to constraints incurred by the type of partial pose measurement system used.

Work by Mooring, Roth, and Driels has determined limits for the number of parameters required to define a complete model [Ref. 18]. Using Denavit-Hartenburg definitions for manipulator kinematics, and including the requirement that any manipulator must be referenced to an external world coordinate frame, four parameters are required for each revolute joint (R) and two for each prismatic joint (P). With the additional six parameters required to ensure independent location of the end effector frame, equation (34) defines the required number of independent parameters, N, for completeness of any manipulator.

$$N = 2P + 4R + 6 \tag{34}$$

For the PUMA and 5R1P robots, N is equal to 30 and 28, respectively. Since any given partial pose measurement system will impose constraints that result in unidentifiable parameters, the value N becomes a maximum value. Therefore, for any six degree of freedom robot, there can be in no case more than 30 parameters in the complete, closed-chain model. Knowing that the measurement system can result in dependencies in the measurement and end effector transformations, T_M and T_E, the total number of identifiable parameters associated only with the manipulator is given by equation (35) [Ref. 19].

$$K = 2P + 4R - 6 \tag{35}$$

For the PUMA, the number of identifiable parameters in the manipulator kinematic model is 18, 22 for the 5R1P.

The value K defines the number of parameters required for the manipulator to move from the manipulator's base frame to the last link, frame *n-1*. The internal base frame and frame *n-1* are unique for any given manipulator and set of joint angles. Therefore the shortest way to close the chain is by an Euler frame-to-frame transformation between the two frames. Given that the manipulator kinematics are unique, there can be no additional dependencies. Therefore, all six Euler parameters are independent. This sets the lower bound on the number of independent, and thus identifiable, parameters for a closed-chain model, equation (36):

$$M = 2P + 4R \tag{36}$$

Thus the number of identifiable parameters in a closed-chain model, *n*, with any measurement system, is governed by equation (37):

$$M \leq n \geq N \tag{37}$$

Once limits to the number of parameters required for a complete model are known, the field of view required to determine the complete model is narrowed. However, equation (37) provides no information regarding the specific number or type of parameters within those limits. It is understood that additional parameter dependencies and, therefore, identifiability, are affected by the type of motion constraint applied to the end effector, and the form of the measurement data (length, azimuth, and elevation etc.) recorded by the system. Determining the relationship between the end effector motion constraints, measured data, and parameter dependencies is the challenge in closed-chain manipulator modeling that often results in a trial and error process to determine the complete model. Attempts to generalize and classify these relationships in this work proved inconclusive. The following section investigates a method to circumvent these problems and arrive at a complete model for any closed-chain system regardless of relationships between the end effector motion constraints, measured data and the identifiable kinematic parameters.

C. GENERAL METHOD

The identification Jacobian for a complete kinematic model is a full rank matrix where the rank equals the number of identifiable parameters [Ref. 18]. Since a complete model contains only identifiable parameters, the rank behavior of the identification Jacobian is studied in detail. The rank of a matrix in a matrix equation, as in equation (22) repeated here, provides an indication of the number of independent equations within the system of equations represented in matrix form.

$$\delta P = J \delta K \tag{38}$$

Therefore, it is assumed that if the rank of the identification Jacobian is less than the number of kinematic parameters (the number of elements in δK), then the model is over-specified and dependencies exist. From Chapter II, Section B-2, the Jacobian is a $p \times n$ matrix, where n is the number of columns and equal to the number of parameters in the model. Each column contains the coefficients for each respective differential kinematic parameter element in the vector δK. Consequently, in the case of an over-specified model, setting the column that multiplies a dependent parameter in the δK vector equal to zero will not change the rank of the Jacobian. Likewise, removing a dependent parameter from δK and its associated column in the Jacobian will not change the rank of J.

An over-specified, closed-chain model of a system is generated in this work, then the identification Jacobian rank behavior is investigated. If the matrix is not full rank then parameter dependencies exist. Those parameters (and their associated columns in the Jacobian), that, once removed, do not affect the rank of J, are then removed from the model. This is repeated until a full rank Jacobian is achieved. The resulting rank matrix will contain only those parameters that are identifiable and define a complete model.

Since equation (34) defines the upper limit of the number of parameters in a complete model, the value N is used to define the over-specified model and parameter set. Once defined, the complete model determination process. In the case of the PUMA manipulator, this is the full 30 parameter model defined in Chapter II.

D. GENERAL PARAMETER REDUCTION ALGORITHM

1. Simulation

One of the by-products of the optimization routine ZXSSQ used in the calibration process is a numerical approximation of the identification Jacobian. By running the first three steps of the calibration simulation (programs JOINT, POSE, and ID6), a Jacobian approximation can be extracted.

In order to determine the complete model parameter set for a six degree of freedom manipulator, the programs POSE and ID6 are written to use the over-specified 30 parameter model for the PUMA manipulator defined in Chapter II.

Programs JOINT and ID6 are modified to represent the particular measurement system used in the calibration process. The end effector pose constraints imposed by the measurement system are accounted for when generating the random joint variable data in program JOINT. The error function in ID6 is modified for each system to reflect the error in only the parameters measured.

Although program ID6 will not converge to an accurate solution of the actual kinematic parameters when using an over-specified model, the Jacobian can be extracted for use in the general parameter reduction program GPRED. Upon extraction of the Jacobian from the routine ZXSSQ, the Jacobian transpose product, J^TJ, is calculated and written to the data file JTJ.DAT. This forms an N x N square matrix that saves memory space and time, independant of the number of pose measurements simulated.

2. Program GPRED

The general parameter reduction program GPRED reads the J^TJ matrix from JTJ.DAT and calculates the rank. From the difference between the rank, R, and the J^TJ matrix dimension N (N is equal to 30 for the PUMA), the number of dependencies and, therefore, non-identifiable parameters is known, equation (39):

$$d = N - R \tag{39}$$

With GPRED using the J^TJ product, a row and a column need to be removed from the matrix to remove a parameter from the model.

GPRED then systematically investigates the dependency of each parameter by removing the associated row and column from J^TJ and recalculating the rank. If the rank is reduced by one, then the parameter removed is independent and identifiable. GPRED then steps to the next parameter and tests it for dependency. If the rank does not change then the parameter is dependent and not identifiable. The process is then sequentially repeated with each successive reduced Jacobian until a full rank matrix is achieved and *d* non-identifiable parameters have been extracted. The parameters remaining in the full rank reduced J^TJ matrix define a complete model, kinematic parameter set. GPRED then steps back out of the reduction and investigates for other possible complete model parameter sets.

The resultant non-identifiable parameter set vectors, associated with each complete model parameter set, are written to the file DDEP.DAT. Note that, for multiple parameter sets, GPRED will find all possible permutations of each parameter set. As a point of reference, this process will be described as the parameter reduction process for the rest of this work.

3. Complete Model Determination of Over-Specified 5R1P Manipulator

In order to demonstrate the use of the complete model determination method described in the previous sections, the parameter reduction process was applied to the

5R1P manipulator with a full pose measurement system. Recall that the 5R1P is a 28 parameter model with one prismatic joint.

Starting with a 30 parameter over-specified model (the 30 parameter PUMA model from Chapter II), the parameter reduction should result in a 28 parameter set that would define a complete model. To do this, the identification simulation routines (programs JOINT and ID6) need to represent the prismatic joint. The 30 parameter PUMA models in the identification routines were modified by defining the translation parameter d_3 as the joint variable, vice θ_3 in the PUMA case, to accommodate the prismatic joint three.

Running this 30 parameter over-specified model through the parameter reduction process revealed a number of sets of dependent, non-identifiable parameters. Of the five parameters listed in Table 3, any pair combination will define a set of non-identifiable parameters. These parameters could be removed from the 30 parameter over-specified model to form a 28 parameter complete model of the 5R1P manipulator.

d_2	a_2	d_3	a_3	d_4

Table 3. 5R1P non-identifiable parameters

Table 3 shows that possible dependencies exist with the link length, a_n, and distance between links, d_n, for joints two and three and the distance between links for joint four. Previously, for a prismatic joint, n, the parameters a_n and d_{n+1} are known to be non-identifiable. For the 5R1P manipulator, this corresponds to a_3 and d_4. The general parameter reduction process confirmed the dependency of parameters a_3 and d_4, and also demonstrated that this set of parameters is not unique. Consequently, any given complete model parameter set for a manipulator is not unique.

The parameter reduction process uncovered all possible permutations of the non-identifiable sets, indicating that the order of parameter removal, within the imbedded loops of the reduction process, has no affect on the outcome.

IV. VERIFICATION OF COMPLETE MODEL DETERMINATION

As previously stated, standardized complete models of full pose measurement systems exist. Determining the complete model for a partial pose measurement system is a different task. This chapter verifies the use of the general complete model on two partial pose, constrained measurement calibration systems, the modified linear slide [Ref. 8] and the fixed length ball bar [Ref. 6].

A. MODIFIED LINEAR SLIDE

1. Physical Description

The linear slide system was used to calibrate a PUMA 560 by Potter [Ref. 7] then modified by Swayze [Ref. 8] to allow for a greater range of joint motion during calibration.

The linear slide, pictured in Figure 18, used the x axis of a coordinate measuring machine (CMM) to accurately measure the distance along the axis from some zero reference point. The original linear slide, used by Potter, bolted the tool flange of the PUMA to a flange on the CMM carriage. This fixed the orientation of the tool frame with respect to the measurement system axis. Swayze modified this system by adding an off-set ball and socket joint between the tool flange and the CMM carriage. This modification allowed for a greater range of motion over the original linear slide, thereby increasing the overall calibration accuracy.

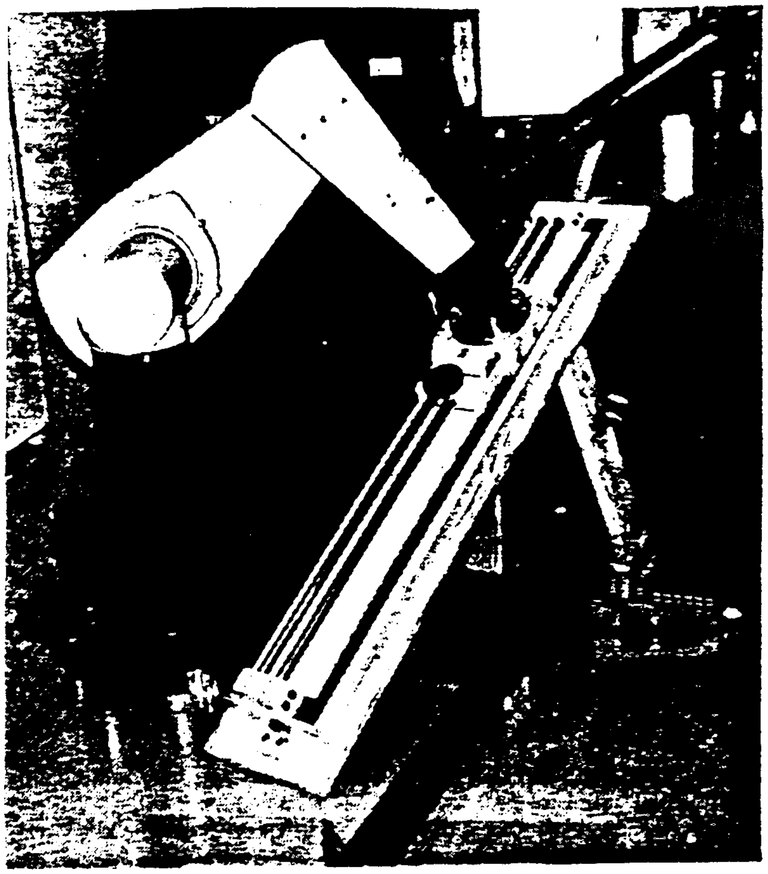

Figure 18. PUMA with linear slide constrained measurement system.

This system places a physical constraint on the end effector, only allowing it to move along the axis defined by the CMM. It measures the position of the end effector frame on that axis.

2. Closed-chain kinematic model

As noted previously, the 18 kinematic parameters of the PUMA manipulator, from the base frame to frame five, are unique and unaffected by different measurement systems. The manipulator kinematic parameters are listed in Table 1. Parameters of the measurement system transformation, T_m^B , and the end effector transformation, T_5^e, must be determined.

For the end effector transformation, the ball joint provides three degrees of freedom. This allows rotation about the three axes of a frame placed at the center of the ball. The free rotation and fixed translation of the ball joint, therefore, define a point in space. The end effector frame is placed at this point, but the orientation of the frame is undefined.

The end effector transformation, therefore, moves from frame (5) to a point in space. Any three of the six Euler parameters are necessary to make this transformation, but one parameter must be a translation. Otherwise displacement of the origin is not possible.

As the end effector and carriage assembly move along the CMM track, they define an axis, not just a direction. The origin of this axis is located at the zero point for the CMM readout. This zero point is the origin of the measurement system coordinate frame.

The measurement system transformation is developed by considering the base to world transformation, T_B^M, then inverting it. A three parameter transformation is required to locate the origin at a point in space. Two rotations are then required to align any one axis of the frame with the CMM track. This results in a five parameter transformation. Table 4 is the parameter list, developed by Swayze [Ref. 19], for the two transformations, pictured in Figure 19. Note that the non-identifiable parameters listed in Table 4 are indicated with a bold zero (**0**). With these eight parameters added to the 18 for the manipulator kinematics, there are 26 identifiable parameters in the complete closed-chain model.

$T_M{}^B$	$T_5{}^E$
ϕ_M	ϕ_E
θ_M	0
0	0
x_M	x_E
y_M	0
z_M	z_E

Table 4. Identifiable kinematic parameters for the modified linear slide.

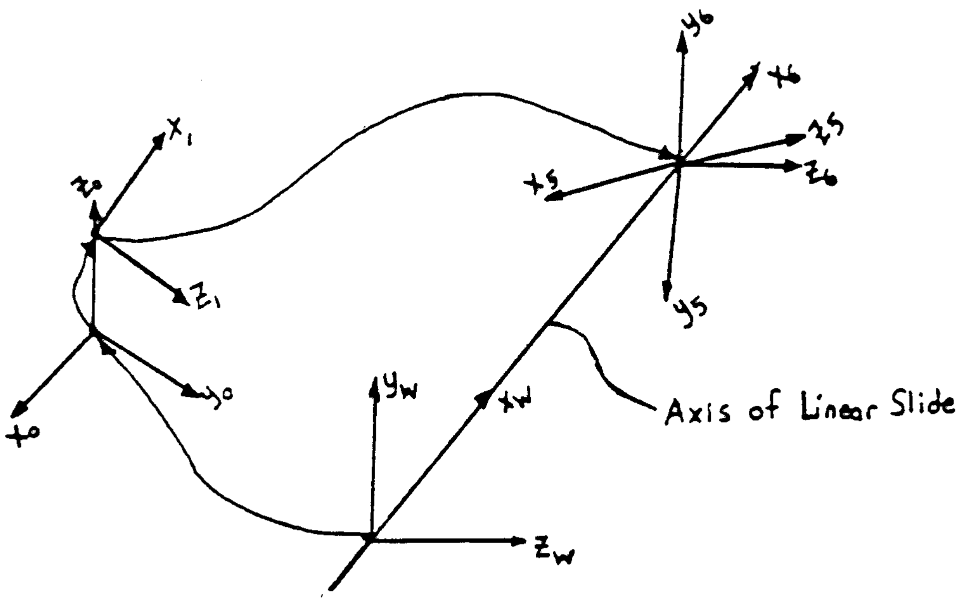

Figure 19. Linear slide transformations.

3. GPRED Results

With the error function properly defined in the simulation programs, a 30-parameter, over-specified model on the modified linear slide was run through the parameter reduction process to determine the complete model. The error function, e, is the length measurement difference along the linear slide axis, represented by equation (40):

$$e = l\,[x^2 + y^2 + z^2]^{\frac{1}{2}} \tag{40}$$

where the measured distance is x, with y and z nominally zero, and l is the readout from the CMM system display of the actual value of x.

The rank of the Jacobian for the over-specified, modified linear slide was 26, confirming a 26 parameter complete model. GPRED then found 48 possible parameter sets (Table 5) that each result in a full ranking matrix, and, therefore, define a complete model. GPRED confirmed the parameter set listed in Table 4. Note Table 5 is a condensed representation of the total set of 48 parameter sets. The 16 end effector parameter sets, when added to the measurement system parameter sets with either ϕ_m, θ_m, or ψ_m removed individually, make up the 48 sets. The x indicators represent identifiable parameters.

In order to confirm that other parameter sets are indeed valid and define a possible complete model, one was chosen arbitrarily for simulation. The simulation code described in Chapter II, Section 2-B was modified to reflect the 26 parameter set partially listed in Table 6:

The calibration simulation using the parameters in Table 6, with no noise, converged to within $1.0E^{-3}$ of the actual kinematic parameter values in the simulation. This result confirms that a given complete model for a manipulator is not unique, and that any of the resulting parameter sets could be used to define a closed-chain, complete model. The types of parameters required for the end effector and measurement system transformations were consistent with those discussed in the previous section. Specifically, they were consistent with a five parameter transformation from the measurement system to the base frame, where one rotation parameter is not identifiable. For the end effector transformation, three parameters were identifiable, with one translation in every parameter set.

ϕ_m	θ_m	ψ_m	x_m	y_m	z_m	ϕ_e	θ_e	ψ_e	x_e	y_e	z_e
	or x	or x	x	x	x				x	x	x
	or x	or x	x	x	x			x	x		x
	or x	or x	x	x	x			x	x	x	
	or x	or x	x	x	x		x			x	x
	or x	or x	x	x	x		x		x	x	
	or x	orx	x	x	x		x	x			x
	or x	orx	x	x	x		x	x	x		
	or x	or x	x	x	x	x				x	x
	or x	or x	x	x	x	x			x		x
	or x	or x	x	x	x	x			x	x	
	or x	or x	x	x	x	x		x			x
	or x	or x	x	x	x	x		x		x	
	or x	or x	x	x	x	x		x	x		
	or x	or x	x	x	x	x	x				x
	or x	or x	x	x	x	x	x			x	
	or x	or x	x	x	x	x	x		x		

Table 5. Complete model parameter sets for modified linear slide

GPRED again returned all possible permutations of every parameter set, proving that the order of parameter removal within the process had no bearing on the end result.

B. FIXED LENGTH BALL BAR

1. Physical Description

The second constrained, partial pose measurement system considered is a ball bar of fixed length, developed by Driels [Ref. 6]. The ball bar constraint system used for manipulator calibration, illustrated in Figure 20, is a rigid bar of fixed, known length with a ball and socket joint at each end. One ball and socket joint is fixed to the robot's work bench, the other is attached to the end flange of the last link of the manipulator. The end effector is then physically constrained to a surface of fixed radial distance from the reference point in the world or measurement system coordinate. The end effector is thus limited to placement in a hemi-spherical surface, with a radius equal to the length of the bar.

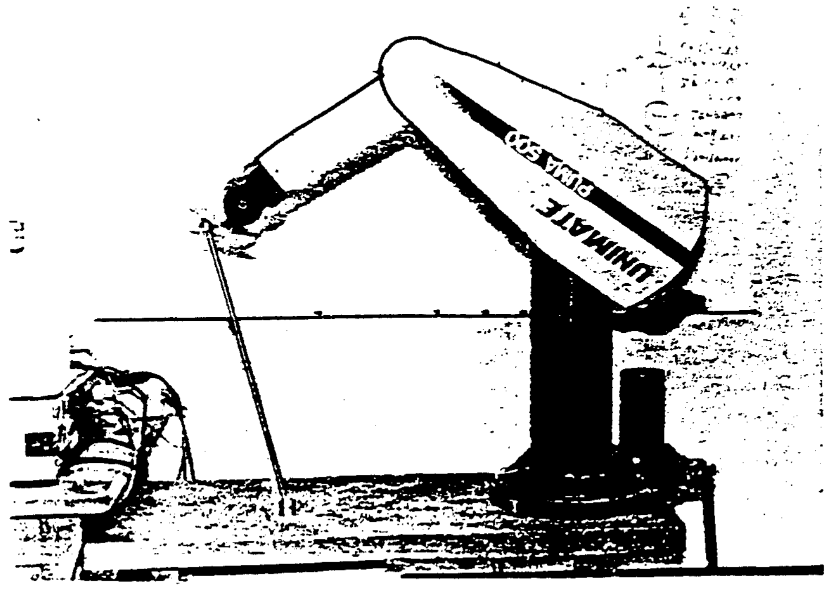

Figure 20. PUMA with ball bar constrained measurement system.

T_M^B	T_5^E
0	0
θ_M	0
ψ_M	0
x_M	x_E
y_M	y_E
z_M	z_E

Table 6. Verified parameter set for the modified linear slide.

2. Closed-Chain Model

As with the modified linear slide, the 18 parameters associated with the PUMA manipulator are unchanged. The measurement system and end effector frame placement, and transformations, follow a similar development to the ball joint end effector of the modified linear slide.

The location of the measurement system coordinate frame is chosen to be at the center of the ball at the lower end of the bar. As with the modified linear slide end effector, the ball joint defines a point in space with no frame orientation information. Again, the transform from the base frame to the measurement system is considered, then inverted. This is a frame to a point transformation, thus requiring three parameters, one of which must be a translation. The measurement system frame is chosen to have the same orientation as the base frame, therefore, three translation parameters, x_m,y_m, and z_m, comprise the identifiable parameters in the measurement system transformation.

The end effector transformation is also a frame to a point. The ball joint at the end effector is displaced from the manipulator's frame five axis, z_5, in order to allow identification of the encoder off-set, or joint variable, for joint six, $\delta\theta_6$ (refer to Figure 21).

The three parameters ϕ_6, x_6, and z_6, are chosen for the frame-to-point transformation. The parameter ϕ_6 was chosen as it inherently includes both the joint variable for joint six, and the constant off-set to account for the mounting plate orientation on the PUMA's end flange. The measurement system and end effector frame transformation parameters are listed in Table 7.

T_M^B	T_5^E
0	ϕ_E
0	0
0	0
x_M	x_E
y_M	0
z_M	z_E

Table 7. Ball bar complete closed-chain model parameters.

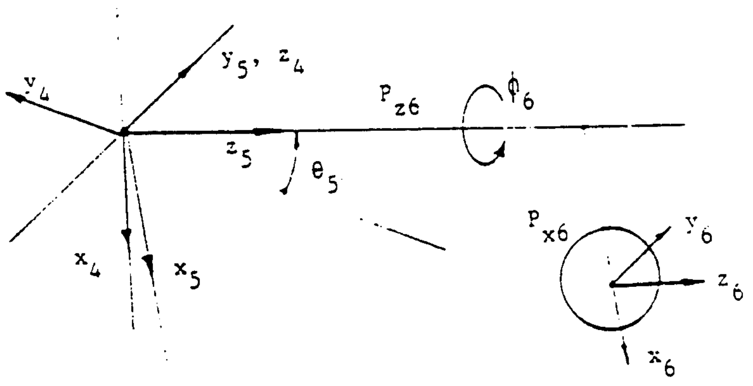

Figure 21. Ball bar end effector transformation.

3. GPRED Results

Using the general parameter reduction process, possible complete parameter sets were determined from the 30 parameter, over-specified model. Before the calibration simulation and parameter reduction could be performed, the appropriate error function was defined. The error function for the ball bar system is similar to the linear slide, repeated here and illustrated in Figure 22.

$$e = |d - l| \tag{41}$$

$$d = [x^2 + y^2 + z^2]^{\frac{1}{2}} \tag{42}$$

where the value *l* is the fixed length of the bar, and *d* is the distance between the end effector and the measurement system coordinate frame.

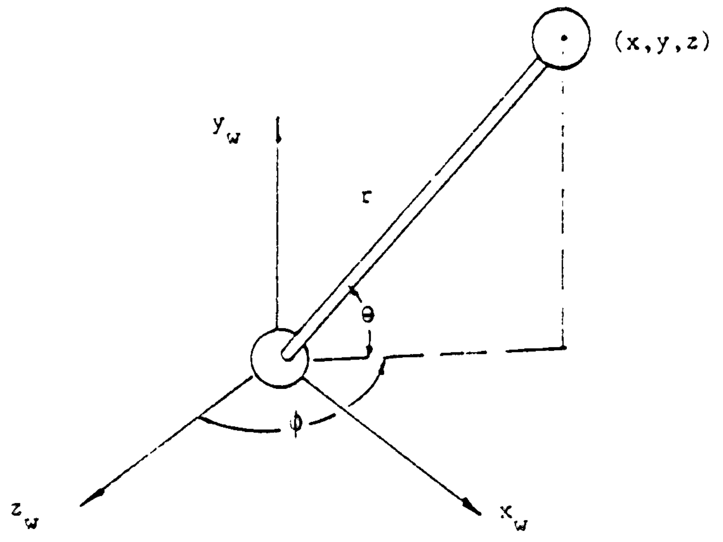

Figure 22. Ball bar kinematics.

From GPRED, the rank of the over-specified model was 24, confirming that the complete model has 24 parameters. The output from GPRED showed that 17 possible parameter sets exist, Table 8, to include the identifiable parameter set developed by Driels

[Ref. 6]. The *x* markers in Table 8 indicate an identifiable parameter, with only the measurement system and end effector transformations shown, since no change occurred within the manipulator kinematic parameter set.

ϕ_m	θ_m	ψ_m	x_m	y_m	z_m	ϕ_e	θ_e	ψ_e	x_e	y_e	z_e
			x	x	x				x	x	x
			x	x	x			x	x		x
			x	x	x			x	x	x	
			x	x	x		x			x	x
			x	x	x		x		x		x
			x	x	x		x		x	x	
			x	x	x		x	x			x
			x	x	x		x	x	x		
			x	x	x	x				x	x
			x	x	x	x			x		x
			x	x	x	x			x	x	
			x	x	x	x		x			x
			x	x	x	x		x		x	
			x	x	x	x		x	x		
			x	x	x	x	x				x
			x	x	x	x	x			x	
			x	x	x	x	x		x		

Table 8. Complete model parameter sets for fixed length ball bar.

The resulting parameter sets reveal that no change occurred in the measurement system kinematics, only the three translation parameters are identifiable. The end effector transformation followed along the same line as the modified linear slide end effector, requiring any three parameters, as long as one of the three is a translation. The parameter set in Table 9 was arbitrarily chosen in order to confirm that the other parameter sets actually do converge to an accurate solution of the actual kinematic parameters.

The simulation programs were modified by removing the effect of the non-identifiable parameters listed in Table 9, resulting in a 24 parameter model. The results of this 24 parameter simulation, with no noise injected, converged to within 1.0E^{-3} of the actual parameters. This confirms that the parameters listed in Table 9, added to the 18 manipulator kinematic parameters, define a complete model of the PUMA with a ball bar constrained measurement system. Consequently, the other parameter sets listed in Table 8 will also define other non-unique complete model parameter sets.

T_M^B	T_5^E
0	ϕ_E
0	0
0	0
x_M	x_E
y_M	0
z_M	z_E

Table 9. Verified parameter set for the fixed length ball bar.

V. DISCUSSION OF RESULTS

The general parameter reduction routine, GPRED, is able to determine complete model kinematic parameter sets by examining the identification Jacobian produced by the calibration simulation of an over-specified model. This was demonstrated by applying the process to the Driels and Pathre 5R1P manipulator and the PUMA 560 manipulator with two different partial pose, constrained measurement systems - the modified linear slide, and the fixed length ball bar. The results indicate that by starting with the 30 parameter over-specified model used in this work, the parameter reduction process can determine the complete, closed-chain kinematic model parameter set for any six degree of freedom mechanical linkage manipulator and measurement system, full or partial pose. The following explanation outlines how to use this method to determine the complete model kinematic parameter set.

The first step in the process is to appropriately define the error function to be used in the simulation code. This error function is based on the actual type of measurement taken and the parameters in the world coordinate frame that relate to that measurement. Using the ball bar as an example, once the world coordinate frame is fixed to the lower end of the ball bar, the *x,y,z* coordinates of the other end are used to calculate a radial distance to that point. This distance is then compared to the length of the bar to determine the error (refer to equation 41). The *x,y,z* parameter values are determined from the forward kinematic solution transformation matrix, for the given set of joint angles, to calculate the radial distance. The error function in the subroutine PUMA_ARM in ID6 can then be modified to represent the measurement system employed.

If the measurement system constrains the motion of the end effector, then the error function, in the program JOINT, must be modified in a similar way. The program JOINT generates the random joint data used in the simulation. The error function is used to ensure the joint data generated actually places the end effector within the constraints.

If the manipulator under study incorporates a prismatic joint, as opposed to all revolute joints, then the joint variable definitions need to be modified to reflect the change. This is a simple, two-line change in each forward kinematic solution calculation, for which there is one in the program POSE and one in the subroutine PUMA_ARM. The program ID6P30, used to simulate the 5R1P, provides an adequate example.

If other than a six degree of freedom manipulator is being used, the standard Denavit-Hartenburg and Modified Denavit-Hartenburg transformations can be used to generate the over-specified model. The maximum number of parameters should not exceed the value of N from equation (43), where *n* is the total number of joints.

$$N = 4n + 6 \qquad (43)$$

The forward kinematic solutions would need to be modified to represent this manipulator.

Once appropriate changes have been made to the simulation code, the simulation is performed, followed by the general parameter reduction routine GPRED. Appendix A contains GPRED for a 30 parameter over-specified model and ten levels of reduction. This satisfies the parameter limits described in equation (37).

GPRED outputs a series of vectors of non-identifiable parameters. Each set, when removed from the over-specified model, represents a non-unique model parameter set. The vectors contain the index values corresponding to the parameter vector element assignment in the program ID6. GPRED writes all the permutations of the dependent non-identifiable parameter vectors to DEPP.DAT.

Having completed this process, any of the resulting parameter sets can be implemented to conduct the actual manipulator calibration. The number and types of observations, or measurements, needed to achieve an accurate calibration are addressed elsewhere.

Although the simulation code allows noise to be injected into the simulated system, noise was not required to observe the Jacobian rank behavior of the systems investigated in this work. The effect of noise would require many more measurements to be simulated to prevent a significant reduction to the final accuracy. Noise effects on robot calibration and simulation have been studied elsewhere in depth. The only advantage to injecting noise in the system would be to determine whether the simulation phase (in order to determine the Jacobian approximation) of the general complete model determination process could be avoided. Instead experimental data may be used directly as an input to program ID6, and the general parameter reduction, GPRED, resulting in complete model parameter sets without required simulation. The new, complete model could then be implemented and the manipulator calibrated using the same data.

VI. CONCLUSIONS

The results of the research reported in this thesis provide the following conclusions:

1. A general method to determine a complete closed-chain model of any manipulator and measurement system, full or partial pose, beginning with an over-specified model, exists and works successfully.

2. Any given, complete model parameter set for a robot manipulator is not unique.

3. The identification Jacobian rank behavior directly correlates to parameter dependence and identifiability.

4. The order of dependent parameter removal from the model has no bearing on the outcome of the final model.

5. Changes in the manipulator configuration only affect the manipulator kinematics.

6. Changes in the measurement system configuration only affect the measurement system kinematics.

APPENDIX A. PROGRAM GPRED

```
c ******************************************

        program gpred

c This program reduces a 30 parameter model pseudo
c inverse jacobian (jTj) by each parameter and investigates the
c changes in the rank of the reduced matrix IOT determine
c the minimum set of independent parameters

        INTEGER NN, NDP
        parameter(NN = 30, NDP = 30)

        REAL*8 JTJ(NN,NN), JR1(NN-1,NN-1), JR2(NN-2,NN-2)
        REAL*8 JR3(NN-3,NN-3), JR4(NN-4,NN-4), JR5(NN-5,NN-5)
        REAL*8 JR6(NN-6,NN-6), JR7(NN-7,NN-7), JR8(NN-8,NN-8)
        REAL*8 JR9(NN-9,NN-9), JR10(NN-10,NN-10)

        INTEGER DP(NDP), DPI(NDP),DPF(NDP)

        REAL*8 U(NN,NN), V(NN,NN), TOL, S(NN)
        REAL*8 U1(NN-1,NN-1), V1(NN-1,NN-1), S1(NN-1)
        REAL*8 U2(NN-2,NN-2), V2(NN-2,NN-2), S2(NN-2)
        REAL*8 U3(NN-3,NN-3), V3(NN-3,NN-3), S3(NN-3)
        REAL*8 U4(NN-4,NN-4), V4(NN-4,NN-4), S4(NN-4)
        REAL*8 U5(NN-5,NN-5), V5(NN-5,NN-5), S5(NN-5)
        REAL*8 U6(NN-6,NN-6), V6(NN-6,NN-6), S6(NN-6)
        REAL*8 U7(NN-7,NN-7), V7(NN-7,NN-7), S7(NN-7)
        REAL*8 U8(NN-8,NN-8), V8(NN-8,NN-8), S8(NN-8)
        REAL*8 U9(NN-9,NN-9), V9(NN-9,NN-9), S9(NN-9)
        REAL*8 U10(NN-10,NN-10), V10(NN-10,NN-10), S10(NN-10)


        INTEGER SIZE, SIZE1, SIZE2, RJAC, N, RJTJ
        INTEGER R1, R2, R3, R4, R5, R6, R7, R8, R9, R10
        INTEGER Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8, Z9, Z10
        INTEGER IP, LVL, COUNT, D
        INTEGER I,II,III,JJ,JJJ,K,m

C write jTj to a file in column order

        OPEN (10,NAME='DEPP_1s.DAT', STATUS='NEW')
        OPEN (15,NAME='JTJ.DAT', STATUS='OLD')

        read(15,*)
        read(15,*)n,rjtj

        read(15,*)
        read(15,*)

        do k = 1,N
           do kk = 1,N
                read(15,*) jtj(kk,k)
           enddo
        enddo

        CLOSE (15)

        IP = 00
        TOL = 1.E-3
```

```
      Z1=NN-1
      Z2 = Z1-1
      Z3 = Z2-1
      Z4 = Z3-1
      Z5 = Z4-1
      Z6 = Z5-1
      Z7 = Z6-1
      Z8 = Z7-1
      Z9 = Z8-1
      Z10 = Z9-1

      COUNT = 1
      D = N-RJTJ
      m = 1

      CALL DLSVRR(NN, NN, JTJ, NN, IP, TOL, RJAC, S, U, NN, V, NN)

      IF (RJAC .LT. N .AND. RJTJ .EQ. RJAC) THEN

       DO I = 1,NN
        WRITE(6,*) '1-',I,count,m
        m = m+1
        CALL MARED(JTJ,I,N,JR1)
        CALL DLSVRR(Z1,Z1,JR1,NN-1,IP, TOL,R1,
     &                 S1,U1,NN-1,V1,NN-1)

        IF (R1 .EQ. RJAC) THEN

         LVL = 1
         DP(LVL) = I
         IF (D .EQ. LVL) THEN
          CALL SORTDP(DP,D)
          COUNT = COUNT+1
          GOTO 100
         END IF

       DO II = 1,Z1

        WRITE(6,*) '2-',II,count,m
        m = m+1
        CALL MARED(JR1,II,Z1,JR2)
        CALL DLSVRR(Z2,Z2,JR2,NN-2,IP, TOL,R2,
     &                 S2,U2,NN-2,V2,NN-2)

        IF (R2 .EQ. RJAC) THEN

         LVL = 2
         DP(LVL) = II
         IF (D .EQ. LVL) THEN
          CALL SORTDP(DP,D)
          COUNT = COUNT+1
          GOTO 200
         END IF

       DO III = 1,Z2

        WRITE(6,*) '3-',III,count,m
        m = m+1
        CALL MARED(JR2,III,Z2,JR3)
```

```
          CALL DLSVRR(Z3,Z3,JR3,NN-3,IP, TOL,R3,
     &                 S3,U3,NN-3,V3,NN-3)

          IF (R3 .EQ. RJAC) THEN

           LVL = 3
           DP(LVL) = III
           IF (D .EQ. LVL) THEN
            CALL SORTDP(DP,D)
            COUNT = COUNT+1
            GOTO 300
           END IF

         DO JJ = 1,Z3

          WRITE(6,*) '4-',JJ,count,m
          m = m+1
          CALL MARED(JR3,JJ,Z3,JR4)
          CALL DLSVRR(Z4,Z4,JR4,NN-4,IP, TOL,R4,
     &                 S4,U4,NN-4,V4,NN-4)

          IF (R4 .EQ. RJAC) THEN

           LVL = 4
           DP(LVL) = JJ
           IF (D .EQ. LVL) THEN
            CALL SORTDP(DP,D)
            COUNT = COUNT+1
            GOTO 400
           END IF

         DO JJJ = 1,Z4

          WRITE(6,*) '5-',JJJ,count,m
          m = m+1
          CALL MARED(JR4,JJJ,Z4,JR5)
          CALL DLSVRR(Z5,Z5,JR5,NN-5,IP, TOL,R5,
     &                 S5,U5,NN-5,V5,NN-5)

          IF (R5 .EQ. RJAC) THEN

           LVL = 5
           DP(LVL) = JJJ
           IF (D .EQ. LVL) THEN
            CALL SORTDP(DP,D)
            COUNT = COUNT+1
            GOTO 500
           END IF

         DO JII = 1,Z5

          WRITE(6,*) '6-',JII,count,m
          m = m+1
          CALL MARED(JR5,JII,Z5,JR6)
          CALL DLSVRR(Z6,Z6,JR6,NN-6,IP, TOL,R6,
     &                 S6,U6,NN-6,V6,NN-6)

          IF (R6 .EQ. RJAC) THEN

           LVL = 6
          DP(LVL) = JII
          IF (D .EQ. LVL) THEN
           CALL SORTDP(DP,D)
           COUNT = COUNT+1
           GOTO 600
          END IF

      DO JJI = 1,Z6

        WRITE(6,*) '7-',JJI,count,m
        m = m+1
        CALL MARED(JR6,JJI,Z6,JR7)
        CALL DLSVRR(Z7,Z7,JR7,NN-7,IP, TOL,R7,
&                      S7,U7,NN-7,V7,NN-7)

        IF (R7 .EQ. RJAC) THEN

          LVL = 7
          DP(LVL) = JJI
          IF (D .EQ. LVL) THEN
           CALL SORTDP(DP,D)
           COUNT = COUNT+1
           GOTO 700
          END IF

      DO JIJ = 1,Z7

        WRITE(6,*) '8-',JIJ,count,m
        m = m+1
        CALL MARED(JR7,JIJ,Z7,JR8)
        CALL DLSVRR(Z8,Z8,JR8,NN-8,IP, TOL,R8,
&                      S8,U8,NN-8,V8,NN-8)

        IF (R8 .EQ. RJAC) THEN

          LVL = 8
          DP(LVL) = JIJ
          IF (D .EQ. LVL) THEN
           CALL SORTDP(DP,D)
           COUNT = COUNT+1
           GOTO 800
          END IF

      DO IIJ = 1,Z8

        WRITE(6,*) '9-',IIJ,count,m
        m = m+1
        CALL MARED(JR8,IIJ,Z8,JR9)
        CALL DLSVRR(Z9,Z9,JR9,NN-9,IP, TOL,R9,
&                      S9,U9,NN-9,V9,NN-9)

        IF (R9 .EQ. RJAC) THEN

          LVL = 9
          DP(LVL) = IIJ
          IF (D .EQ. LVL) THEN
           CALL SORTDP(DP,D)
           COUNT = COUNT+1
           GOTO 900
          END IF
```

```
         DO IJJ = 1,Z9

          WRITE(6,*) '10-',IJJ,count,m
          m = m+1
          CALL MARED(JR9,IJJ,Z9,JR10)
          CALL DLSVRR(Z10,Z10,JR10,NN-10,IP, TOL,R10,
     &                      S10,U10,NN-10,V10,NN-10)

          IF (R10 .EQ. RJAC) THEN

           LVL = 10
           DP(LVL) = IJJ
           IF (D .EQ. LVL) THEN
            CALL SORTDP(DP,D)
            COUNT = COUNT+1
            GOTO 1000
           END IF

1000       END IF
          ENDDO
900       END IF
          ENDDO
800       END IF
          ENDDO
700       END IF
          ENDDO
600       END IF
          ENDDO
500       END IF
          ENDDO
400       END IF
          ENDDO
300       END IF
          ENDDO
200       END IF
          ENDDO
100       END IF
          ENDDO

        END IF

        WRITE(10,*) '# parameter vectors =',m

        END

C *************************************************

        SUBROUTINE SORTDP (DP,DIF)

        INTEGER DIF, K, I, J, ss, s
        INTEGER DP(DIF), DPI(10), DPF(10)
        parameter(ss = 30)
        integer pl(ss), rpl(ss-1)

        do i = 1,ss
           pl(i) = i
        enddo

        s = ss
        DO J = 1,DIF
           call vred (dp(j),pl,s,rpl,dpi(j))
           s = s-1
           do k = 1,s
                pl(k) = rpl(k)
           enddo
        ENDDO
        if (dp(1) .ne. dpi(1)) then
           write(6,*) 'subroutine sort misfunction'
        end if

        CALL SVIGN (DIF,DPI,DPF)
        WRITE(10,*) (DPF(JK), JK=1,DIF)
        WRITE(6,*) (DP(JK), JK=1,DIF)
        write(6,*) (dpi(jk), jk=1,dif)
        WRITE(6,*) (DPF(JK), JK=1,DIF)
        RETURN
        END

C **********************************************

        subroutine mared(JT,IN,S,JNEW)

        integer S, IN, JR, JC, KR, KC
        real*8 JT(S,S), JNEW(S-1,S-1)

        JR = 1
        DO KR = 1,S-1
             IF (KR.EQ.IN) THEN
                JR = JR+1
             END IF
           JC = 1
           DO KC = 1,S-1
                IF (KC.EQ.IN) THEN
                  JC = JC+1
                END IF

                JNEW(KR,KC) = JT(JR,JC)
                JC = JC+1

           ENDDO

           JR = JR+1
        ENDDO
C       WRITE(6,*) JT(1,1),JNEW(1,1)
        RETURN
        END

c **********************************************

        subroutine vred(in,pli,s,plo,pv)

        integer in, s, pv, jc, kc
        integer pli(s), plo(s-1)


        pv = pli(in)
        JC = 1
        DO KC = 1,S-1
                IF (KC.EQ.IN) THEN
```

```
                  JC = JC+1
                END IF

                plo(KC) = pli(JC)
                JC = JC+1

        ENDDO
        return
        end
```

APPENDIX B. PROGRAM POSE FOR LINEAR SLIDE

```
        PROGRAM blinsc

C  This program generates joint angles for the Puma manipulator
C  arm.  It presumes that the tool frame of the manipulator is
C  constrained to move in the positive x direction only.  The
C  tool is constrained by a ball joint mounted to a sliding linear scale.
C  The values along the x direction are determined by a random number
C   generator.

        INTEGER LDFJAC, M, N, obs, nobs
        PARAMETER (LDFJAC=3, M=LDFJAC, N=6)

        real*8 fi0, th0, si0, px0, py0, pz0
        REAL*8 DT1, DT2, DT3, DT4, DT5
        REAL*8 DD1, DD2, DD3, DD4, DD5
        REAL*8 AA1, AA2, AA3, AA4, AA5
        REAL*8 AL1, AL2, AL3, AL4, AL5
        REAL*8 BL1, BL2, BL3, BL4, BL5
        REAL*8 DF6, FI6, TH6, SI6, PX6, PY6, PZ6

        REAL*8 RN1,RN2,RN3,RN4,RN5,RN6
        REAL*8 RN7,RN8,RN9,RN10,RN11,RN12
        REAL*8 RN13,RN14,RN15,RN16,RN17,RN18

        INTEGER infer,ier,iopt,nsig,maxfn

        REAL*8 FJAC(LDFJAC,N), xjtj((n+1)*n/2), xjac(ldfjac,n)
        REAL*8 parm(4), f(ldfjac), work((5*n)+(2*m)+((n+1)*n/2))
        REAL*8 X(N)
        REAL*8 magnx,magnl

        EXTERNAL PUMA_ARM

        INTEGER I, J, K, nou
        REAL*8 TDES(3), T(4,4), SCALE, DANGLE, DLENTH, NUM

        COMMON /PDATA/ TDES, DANGLE, DLENTH, T
        COMMON /KIN/ fi0, th0, si0, px0, py0, pz0,
     &                DT1,DT2,DT3,DT4,DT5,
     &                AL1,AL2,AL3,AL4,AL5,
     &                AA1,AA2,AA3,AA4,AA5,
     &                DD1,DD2,DD3,DD4,DD5,
     &                BL1,BL2,BL3,BL4,BL5,
     &                DF6,TH6,SI6,PX6,PY6,PZ6

C Initialize data variables

        obs=0

C Open data files for input

        OPEN (10, NAME='puma-pos.dat', STATUS='NEW')
        OPEN (9, NAME='input.dat', STATUS='OLD')

C Read input kinematic data

        read (9,*)
        read (9,*) fi0,th0,si0,px0,py0,pz0
```

```
      read (9,*) dt1,dd1,aa1,al1,bl1
      read (9,*) dt2,dd2,aa2,al2,bl2
      read (9,*) dt3,dd3,aa3,al3,bl3
      read (9,*) dt4,dd4,aa4,al4,bl4
      read (9,*) dt5,dd5,aa5,al5,bl5
      read (9,*)
      read (9,*) df6,th6,si6,px6,py6,pz6
      read (9,*)
      read (9,*) nobs,nou,dangle,dlenth,magnx,magnl

      close (9)

C Adjust nominal values

      fi0=fi0+dangle
      th0=th0+dangle
      si0=si0+dangle
      px0=px0+dlenth
      py0=py0+dlenth
      pz0=pz0+dlenth

      dt1=0.0
      dt2=dt2+dangle
      dt3=dt3+dangle
      dt4=dt4+dangle
      dt5=dt5+dangle

      al1=al1+dangle
      al2=al2+dangle
      al3=al3+dangle
      al4=al4+dangle
      al5=al5+dangle

      aa1=aa1+dlenth
      aa2=aa2+dlenth
      aa3=aa3+dlenth
      aa4=aa4+dlenth
      aa5=aa5+dlenth

      dd1=0.0
      dd2=0.0
      dd3=dd3+dlenth
      dd4=dd4+dlenth
      dd5=dd5+dlenth

      bl1=bl1
      bl2=bl2+dangle
      bl3=bl3
      bl4=bl4
      bl5=bl5

      df6=df6+dangle
      th6=th6+dangle
      si6=si6+dangle
      px6=px6+dlenth
      py6=py6+dlenth
      pz6=pz6+dlenth

C Get random number seed
```

```
        write (6,*) 'Type in a 6-digit random number seed'
        read (5,*) iseed

C Start of main loop

 1010   obs=obs+1

C Set joint angles to zero

        x(1)=70.0
        x(2)=0.0
        x(3)=90.0
        x(4)=0.0
        x(5)=50.0
        x(6)=90.0

C Get random bar lengths

 1000   call random (iseed,num)
        num=num*940.0

C Establish desired tool point

        TDES(1)= num
        TDES(2)= 0.0
        TDES(3)= 0.0

C Call IMSL ZXSSQ for inverse kinematic solution

        nsig=4
        eps=0.0
        delta=0.0
        maxfn=500
        iopt=1
        ixjac=ldfjac

        CALL ZXSSQ(puma_arm,m,n,nsig,eps,delta,maxfn,iopt,parm,x,
     &                    ssq,f,xjac,ixjac,xjtj,work,infer,ier)

C Check for singularities

        if (ssq .gt. 0.00001) goto 1000

C Print results to 2 decimal places

        write(6,*) obs,ssq

C Generate the random noise

        CALL RANDOM (ISEED,RN1)
        CALL RANDOM (ISEED,RN2)
        CALL RANDOM (ISEED,RN3)
        CALL RANDOM (ISEED,RN4)
        CALL RANDOM (ISEED,RN5)
        CALL RANDOM (ISEED,RN6)
        CALL RANDOM (ISEED,RN7)
        CALL RANDOM (ISEED,RN8)
        CALL RANDOM (ISEED,RN9)
        RN1 =  MAGNX * (2.0 * RN1 - 1.0)
        RN2 =  MAGNX * (2.0 * RN2 - 1.0)
        RN3 =  MAGNX * (2.0 * RN3 - 1.0)

        RN4 =  MAGN1 * (2.0 * RN4 - 1.0)
        RN5 =  MAGN1 * (2.0 * RN5 - 1.0)
        RN6 =  MAGN1 * (2.0 * RN6 - 1.0)
        RN7 =  MAGN1 * (2.0 * RN7 - 1.0)
        RN8 =  MAGN1 * (2.0 * RN8 - 1.0)
        RN9 =  MAGN1 * (2.0 * RN9 - 1.0)


        X(1) = X(1) + RN4
        X(2) = X(2) + RN5
        X(3) = X(3) + RN6
        X(4) = X(4) + RN7
        X(5) = X(5) + RN8
        X(6) = X(6) + RN9

        tdes(1)=tdes(1)+rn1
        tdes(2)=tdes(2)+rn2
        tdes(3)=tdes(3)+rn3

        write (10,*) X(1),X(2),X(3),X(4),X(5),X(6)
        write (10,*) tdes(1),tdes(2),tdes(3)
        write (10,*)

C Continue for other bar angles

        if (obs .lt. nobs) go to 1010

        CLOSE (10)

        END

C ****************************************************************

        SUBROUTINE PUMA_ARM (X,M,N,F)

C This subroutine calculates the non-linear function for the use of
C the IMSL routine ZXSSQ. It is the forward kinematic solution for
C the PUMA manipulator.

        INTEGER M, N
        REAL*8 X(N), F(M)

        INTEGER II, JJ
        real*8 fi0, th0, si0, px0, py0, pz0
        REAL*8 DT1, DT2, DT3, DT4, DT5
        REAL*8 DD1, DD2, DD3, DD4, DD5
        REAL*8 AA1, AA2, AA3, AA4, AA5
        REAL*8 AL1, AL2, AL3, AL4, AL5
        REAL*8 BL1, BL2, BL3, BL4, BL5
        REAL*8 DF6, FI6, TH6, SI6, PX6, PY6, PZ6

        REAL*8 TH1, TH2, TH3, TH4, TH5
        REAL*8 T0(4,4), T1(4,4), T2(4,4), T3(4,4), T4(4,4)
        REAL*8 T5(4,4), T6(4,4), trpy(4,4), txyz(4,4)
        REAL*8 TIMAT(4,4), T(4,4), td(4,4)
```

```
      INTEGER I, J, K
      REAL*8 TDES(3), DANGLE, DLENTH,scale

      COMMON /PDATA/ TDES, DANGLE, DLENTH, T
      COMMON /KIN/ fi0,th0,si0,px0,py0,pz0,
     &                DT1,DT2,DT3,DT4,DT5,
     &                AL1,AL2,AL3,AL4,AL5,
     &                AA1,AA2,AA3,AA4,AA5,
     &                DD1,DD2,DD3,DD4,DD5,
     &                BL1,BL2,BL3,BL4,BL5,
     &                DF6,TH6,SI6,PX6,PY6,PZ6

C Initialize the TIMAT matrix to an I matrix:

      DATA TIMAT/1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1/

      scale=100.0

C Initialize the T matrix to an I matrix

      DO II = 1,4
      DO JJ = 1,4
                T(II,JJ) = TIMAT(II,JJ)
      ENDDO
      ENDDO

C Manipulator joint angles

      TH1 = DT1 + X(1)
      TH2 = DT2 + X(2)
      TH3 = DT3 + X(3)
      TH4 = DT4 + X(4)
      TH5 = DT5 + X(5)
      FI6 = DF6 + X(6)

C Compute the T matrices, T1 thru T6:

      call t3rpy (fi0,th0,si0, trpy)
      call t3xyz (px0,py0,pz0, txyz)
      call matmulc (t0,trpy,txyz)

      CALL TRANSFORM ( AL1, AA1, DD1, TH1, BL1, T1 )
      CALL TRANSFORM ( AL2, AA2, DD2, TH2, BL2, T2 )
      CALL TRANSFORM ( AL3, AA3, DD3, TH3, BL3, T3 )
      CALL TRANSFORM ( AL4, AA4, DD4, TH4, BL4, T4 )
      CALL TRANSFORM ( AL5, AA5, DD5, TH5, BL5, T5 )

      CALL t3rpy ( fi6, th6, si6, trpy )
      CALL T3XYZ ( PX6, PY6, PZ6, txyz )
      CALL matmulc ( t6, trpy, txyz )

C Compute the overall transformation, T:

      CALL MATMULA ( T, T0 )
      CALL MATMULA ( T, T1 )
      CALL MATMULA ( T, T2 )
      CALL MATMULA ( T, T3 )
      CALL MATMULA ( T, T4 )
      CALL MATMULA ( T, T5 )
```

```
        CALL MATMULA ( T, T6 )

C Calculate the function F

        f(1)=t(1,4)-tdes(1)
        f(2)=t(2,4)
        f(3)=t(3,4)

       sum = 0.0
       do i=1,3
        xssq=sum+f(i)
       enddo
C       write (6,*) xssq

        RETURN
        END

C ************************************************************

        SUBROUTINE RANDOM (x,z)

C This subroutine generates random numbers in the range 0-1
C using a supplied seed x, the returned random number being z.

        REAL FM, FX, Z
        INTEGER A, X, I, M
        DATA I/1/

        IF ( I .EQ. 0 ) GO TO 1000
        I=0
        M= 2 ** 20
        FM= M
        A= 2**10 + 3

 1000   X= MOD( A*X ,M)
        FX= X
        Z= FX/ FM

        RETURN
        END
```

APPENDIX C. PROGRAM ID6 FOR LINEAR SLIDE

```
      PROGRAM ID6

C Robot Identification using the Non-linear Least Squares method.
C Simulation data is read for the PUMA manipulator from
C the data file PUMA-POS.DAT

C Change parameter LDFJAC to change the number of observations,
C set LDFJAC = 6 * Number of observations

      INTEGER LDFJAC, MM, M, NN, N, NSIG, MAXFN, IOPT, IXJAC, INFER, IER
      PARAMETER (LDFJAC=3*42, MM=LDFJAC, NN=30)

      REAL*8 FJAC(LDFJAC,NN), XJTJ((NN+1)*NN/2)
      REAL*8 PARM(4), F(LDFJAC), WORK((5*NN)+(2*MM)+((NN+1)*NN/2))
      REAL*8 X(NN)
      EXTERNAL PUMA_ARM

      REAL*8 DANGLE, DLENTH, TQ, DQ, EPS, DELTA, SSQ
      REAL*8 SQERR1, SQERR2

      real*8 fi0,th0,si0,px0,py0,pz0
      REAL*8 DT1, DT2, DT3, DT4, DT5
      REAL*8 DD1, DD2, DD3, DD4, DD5
      REAL*8 AA1, AA2, AA3, AA4, AA5
      REAL*8 AL1, AL2, AL3, AL4, AL5
      REAL*8 BL1, BL2, BL3, BL4, BL5
      REAL*8 DF6, TH6, SI6, PX6, PY6, PZ6, FI6

      REAL*8 U(NN,NN), V(NN,NN), TOL, S(NN), FJTJ(NN,NN)
      INTEGER NB, RJTJ, IPATH

      INTEGER I, J, K, NOBS, MAXNOBS
      PARAMETER (MAXNOBS=360)
      REAL*8 TET1(MAXNOBS), TET2(MAXNOBS), TET3(MAXNOBS)
      REAL*8 TET4(MAXNOBS), TET5(MAXNOBS), TET6(MAXNOBS)
      REAL*8 TM(3,MAXNOBS), SCALE
      COMMON /PDATA/ NOBS, TM, SCALE,
     &          TET1, TET2, TET3, TET4, TET5, TET6

C Open data files for inputs and results

      OPEN (8, NAME='RESULT.DAT', STATUS='NEW')
      OPEN (9, NAME='PUMA-POS.DAT', STATUS='OLD')
      OPEN (10,NAME='INPUT.DAT', STATUS='OLD')

c Read input parameters

      read (10,*)
      read (10,*) fi0,th0,si0,px0,py0,pz0
      read (10,*) dt1,dd1,aa1,al1,bl1
      read (10,*) dt2,dd2,aa2,al2,bl2
      read (10,*) dt3,dd3,aa3,al3,bl3
      read (10,*) dt4,dd4,aa4,al4,bl4
      read (10,*) dt5,dd5,aa5,al5,bl5
      read (10,*)
      read (10,*) df6,th6,si6,px6,py6,pz6
      read (10,*)
      read (10,*) nobs,n,dangle,dlenth,magnx,magnl
```

```
        CLOSE (10)

c       write (6,*) 'enter nobs'
c       read (6,*) nobs

C Initialize data variables

        X(1)=fi0
        X(2)=th0
        X(3)=si0
        X(4)=px0
        x(5)=py0
        x(6)=pz0

        X(7)=AA1
        X(8)=AL1

        X(9)=DT2
        X(10)=AA2
        X(11)=AL2
        X(12)=BL2

        X(13)=DT3
        X(14)=DD3
        X(15)=AA3
        X(16)=AL3

        X(17)=DT4
        X(18)=DD4
        X(19)=AA4
        X(20)=AL4

        X(21)=DT5
        X(22)=DD5
        X(23)=AA5
        X(24)=AL5

        x(25)=df6
        x(26)=th6
        x(27)=si6
        x(28)=px6
        x(29)=py6
        x(30)=pz6

C Read simulated joint data and tool pose

        DO J = 1, NOBS
          READ (9,*) TET1(J), TET2(J), TET3(J), TET4(J), TET5(J), TET6(J)
          read(9,*) TM(1,J), TM(2,J), TM(3,J)
c          read(9,*)
c         READ (9,*)a,b,c, TM(1,J)
c         read(9,*)a,b,c,TM(2,J)
c          read(9,*)a,b,c,TM(3,J)
c          read(9,*)
c          read(9,*)
        ENDDO
        CLOSE (9)

C Initialize scale for the angular rows of the Jacobian
```

```
        SCALE=100.0

C Call IMSL routine for non-linear identification

        NSIG=3
        EPS=0.0
        DELTA=0.0
        MAXFN=1500
        IOPT=1
        IXJAC=LDFJAC
        M=3*NOBS

        CALL ZXSSQ(PUMA_ARM,M,N,NSIG,EPS,DELTA,MAXFN,IOPT,
     &           PARM,X,SSQ,F,FJAC,IXJAC,XJTJ,WORK,INFER,IER)

C Calc the jacobian transpose product and it's rank; rank(jTj)

        NB = N
        IPATH = 00
        TOL = 1.E-6

        CALL DMXTXF(M, N, FJAC, LDFJAC, NB, FJTJ, NN)
C        CALL MARANK(FJTJ, N, N, RJTJ)
        CALL DLSVRR(NN, NN, FJTJ, NN, IPATH, TOL, RJTJ, S, U, NN, V, NN)

        OPEN (15,NAME='JTJ.DAT', STATUS='NEW')
        kk = 0
        k = 0
C write jTj to a file in column order

        write(15,*)'mat size(square)','rank'
        write(15,*)n,rjtj
        write(15,*)
        write(15,*)'jTj in column order'
        do k = 1,N
           do kk = 1,N
                write(15,*) fjtj(kk,k)
           enddo
        enddo
        CLOSE (15)

C Save results to data file

        WRITE (8,*)
        WRITE (8,*) 'fi0, th0, si0, px0, py0, pz0'
        WRITE (8,889) X(1), X(2), X(3), X(4), x(5), x(6)
        WRITE (8,*)
        WRITE (8,*) 'DT1, DD1, AA1, AL1, BL1'
        WRITE (8,888) 0.0, 0.0, X(7), X(8), 0.0
        WRITE (8,*)
        WRITE (8,*) 'DT2, DD2, AA2, AL2, BL2'
        WRITE (8,888) X(9), 0.0, X(10), X(11), X(12)
        WRITE (8,*)
        WRITE (8,*) 'DT3, DD3, AA3, AL3, BL3'
        WRITE (8,888) X(13), X(14), X(15), X(16), 0.0
        WRITE (8,*)
        WRITE (8,*) 'DT4, DD4, AA4, AL4, BL4'
        WRITE (8,888) X(17), X(18), X(19), X(20), 0.0
        WRITE (8,*)
        WRITE (8,*) 'DT5, DD5, AA5, AL5, BL5'
        WRITE (8,888) X(21), X(22), X(23), X(24), 0.0
        WRITE (8,*)
        WRITE (8,*) 'DF6, TH6, SI6, PX6, PY6, PZ6'
        WRITE (8,889) x(25), x(26), x(27), x(28), x(29),  x(30)
888    FORMAT ( 5F12.5 )
889    FORMAT ( 6F12.5 )

C Calculate root mean square error in identification

        TQ = DANGLE
        DQ = DLENTH

C Error in identification (angular parameters)

        SQERR1 =
     &  (FI0+TQ-X(1))**2 +(TH0+TQ-X(2))**2 +
     &  +(DT3+TQ-X(12))**2 +(DT4+TQ-X(16))**2 +(DT5+TQ-X(20))**2
     &  +(AL1+TQ-X(7))**2 +(AL2+TQ-X(10))**2
     &  +(AL3+TQ-X(15))**2 +(AL4+TQ-X(19))**2 +(AL5+TQ-X(23))**2
     &  +(BL2+TQ-X(11))**2 +(DT2+TQ-X(8))**2
     &  +(df6+tq-x(24))**2
        SQERR1 = DSQRT( SQERR1/15 )

C Error in identification (length parameters)

        SQERR2 =
     &  (PX0+DQ-X(3))**2 +(AA1+0.0+DQ-X(6))**2 +(AA2+DQ-X(9))**2
     &  +(AA3+DQ-X(14))**2 +(AA4+DQ-X(18))**2 +(AA5+DQ-X(22))**2
     &  +(PY0+DQ-X(4))**2 +(PZ0+DQ-X(5))**2
     &  +(DD3+DQ-X(13))**2 +(DD4+DQ-X(17))**2 +(DD5+DQ-X(21))**2
     &  +(px6+dq-x(25))**2 +(pz6+dq-x(26))**2
        SQERR2 = DSQRT( SQERR2/11 )

        WRITE (8,*)
        WRITE (8,*) 'RMS PARMS (LENGTH), RMS PARMS (ANGLE),'
        WRITE (8,*) SQERR2, SQERR1,'incorrect'
        WRITE (6,*) 'RMS PARMS (LENGTH), RMS PARMS (ANGLE)'
        WRITE (6,*) SQERR2,SQERR1,'incorrect'


        WRITE (8,*)
        WRITE (8,*) 'INFER, IER,NOBS,NSIG, RANK'
        WRITE (8,*)  INFER, IER,NOBS,NSIG,RJTJ
        WRITE (6,*) 'INFER, IER,NOBS,NSIG, RANK'
        WRITE (6,*)  INFER, IER,NOBS,NSIG,RJTJ

        WRITE (8,*)

        CLOSE (8)

        END

C ****************************************************************

        SUBROUTINE PUMA_ARM (X, M, N, F)

C This subroutine calculates the non-linear function for the use of
C the IMSL routine DUNLSF. It is the forward kinematic solution for
C the PUMA manipulator.
```

```
        INTEGER M, N
        REAL*8 X(N), F(M)

        INTEGER II, JJ

        real*8 fi0,th0,si0,px0,py0,pz0
        REAL*8 DT1, DT2, DT3, DT4, DT5
        REAL*8 DD1, DD2, DD3, DD4, DD5
        REAL*8 AA1, AA2, AA3, AA4, AA5
        REAL*8 AL1, AL2, AL3, AL4, AL5
        REAL*8 BL1, BL2, BL3, BL4, BL5
        REAL*8 FI6, TH6, SI6, PX6, PY6, PZ6, DF6

        REAL*8 TH1, TH2, TH3, TH4, TH5
        REAL*8 T0(4,4), T1(4,4), T2(4,4), T3(4,4), T4(4,4)
        REAL*8 T5(4,4), T6(4,4), TRPY(4,4), TXYZ(4,4)
        REAL*8 TIMAT(4,4), T(4,4)
        REAL*8 TINV(4,4), TMJ(4,4), TDELTA(4,4)

        INTEGER I, J, K, NOBS, MAXNOBS
        PARAMETER (MAXNOBS=360)
        REAL*8 TET1(MAXNOBS), TET2(MAXNOBS), TET3(MAXNOBS)
        REAL*8 TET4(MAXNOBS), TET5(MAXNOBS), TET6(MAXNOBS)
        REAL*8 TM(3,MAXNOBS), SCALE
        COMMON /PDATA/ NOBS, TM, SCALE,
     &         TET1, TET2, TET3, TET4, TET5, TET6

C Initialize the TIMAT matrix to an I matrix:

        DATA TIMAT/1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1/

C Set parameters for the manipulator:

      fi0 = X(1)
      th0 = X(2)
      si0 = X(3)
      px0 = X(4)
      py0 = x(5)
      pz0 = x(6)

      DT1 = 0.0
      DD1 = 0.0
      AA1 = X(7)
      AL1 = X(8)
      BL1 = 0.0

      DT2 = X(9)
      DD2 = 0.0
      AA2 = X(10)
      AL2 = X(11)
      BL2 = X(12)

      DT3 = X(13)
      DD3 = X(14)
      AA3 = X(15)
      AL3 = X(16)
      BL3 = 0.0

      DT4 = X(17)
      DD4 = X(18)
```

```
        AA4 = X(19)
        AL4 = X(20)
        BL4 = 0.0

        DT5 = X(21)
        DD5 = X(22)
        AA5 = X(23)
        AL5 = X(24)
        BL5 = 0.0

        DF6 = x(25)
        TH6 = x(26)
        SI6 = x(27)
        px6 = x(28)
        py6 = x(29)
        pz6 = x(30)

C Loop NOBS times

          K = 0
          DO J = 1, NOBS

C Initialize the T matrix to an I matrix

          DO II = 1,3
          DO JJ = 1,4
                  T(II,JJ) = TIMAT(II,JJ)
          ENDDO
          ENDDO

C Manipulator joint angles

          TH1 = DT1 + TET1(J)
          TH2 = DT2 + TET2(J)
          TH3 = DT3 + TET3(J)
          TH4 = DT4 + TET4(J)
          TH5 = DT5 + TET5(J)
          FI6 = DF6 + TET6(J)

C Compute the T matrices, T1 thru T6:

          call t3rpy(fi0,th0,si0, trpy)
          call t3xyz(px0,py0,pz0, txyz)
          call matmulc(t0,trpy,txyz)

          CALL TRANSFORM ( AL1, AA1, DD1, TH1, BL1, T1 )
          CALL TRANSFORM ( AL2, AA2, DD2, TH2, BL2, T2 )
          CALL TRANSFORM ( AL3, AA3, DD3, TH3, BL3, T3 )
          CALL TRANSFORM ( AL4, AA4, DD4, TH4, BL4, T4 )
          CALL TRANSFORM ( AL5, AA5, DD5, TH5, BL5, T5 )

          CALL T3RPY ( FI6, TH6, SI6, TRPY )
          CALL T3XYZ ( PX6, PY6, PZ6, TXYZ )
          CALL MATMULC (T6, TRPY, TXYZ )

C Compute the overall transformation, T:

          CALL MATMULA ( T, T0 )
          CALL MATMULA ( T, T1 )
          CALL MATMULA ( T, T2 )
        CALL MATMULA ( T, T3 )
        CALL MATMULA ( T, T4 )
        CALL MATMULA ( T, T5 )
        CALL MATMULA ( T, T6 )


        f(k+1)=t(1,4)-tm(1,j)
        f(k+2)=t(2,4)
        f(k+3)=t(3,4)

        k=k+3

C End the do-loop for counter J

        ENDDO

C Write RMS error in F

        XSSQ=0.0
        DO II=1,3*NOBS
          XSSQ=XSSQ+F(II)*F(II)
        ENDDO

        XER=SQRT(XSSQ)
          WRITE(6,*) XER

        RETURN
        END
```

APPENDIX D. PROGRAM POSE FOR BALL BAR

```
C *********************************************************************

        PROGRAM PUMABAR

C This program generates a set of joint angles for tha calibration
C of the PUMA manipulator using a ball bar to constrain the end
C point of the manipulator.

        INTEGER LDFJAC, M, N, obs, nobs
        PARAMETER (LDFJAC=3, M=LDFJAC, N=6)

        REAL*8 DT1, DT2, DT3, DT4, DT5
        REAL*8 DD1, DD2, DD3, DD4, DD5
        REAL*8 AA1, AA2, AA3, AA4, AA5
        REAL*8 AL1, AL2, AL3, AL4, AL5
        REAL*8 BL1, BL2, BL3, BL4, BL5
        REAL*8 DF6, FI6, TH6, SI6, PX6, PY6, PZ6
        REAL*8 fi0, th0, si0, px0, py0, pz0

        INTEGER infer,ier,iopt,nsig,maxfn
        REAL*8 FJAC(LDFJAC,N), xjtj((n+1)*n/2), xjac(ldfjac,n)
        REAL*8 parm(4), f(ldfjac), work((5*n)+(2*m)+((n+1)*n/2))
        REAL*8 X(N)
        real*8 r,phimax,phimin,thetamax,thetamin,phi,theta
        real*8 xb,yb,zb,ssq,rr,magnx,magnl

        EXTERNAL PUMA_ARM

        INTEGER I, J, K
        REAL*8 TDES(4,4), qmax(6), qmin(6), SCALE, DANGLE, DLENTH, NUM
        COMMON /PDATA/ TDES, DANGLE, DLENTH, r
        COMMON /KIN/ DT1,DT2,DT3,DT4,DT5,
     &         AL1,AL2,AL3,AL4,AL5,
     &         AA1,AA2,AA3,AA4,AA5,
     &         DD1,DD2,DD3,DD4,DD5,
     &         BL1,BL2,BL3,BL4,BL5,
     &         fi0,th0,si0,px0,py0,pz0,
     &         DF6,TH6,SI6,PX6,PY6,PZ6

C Joint angle ranges

        data qmin/-160.0,-223.0,-52.0,-110.0,-100.0,-266.0/
        data qmax/160.0, 43.0, 232.0, 170.0, 100.0, 266.0/

C Initialize data variables

        obs=0

C Open data files for input

        OPEN (10, NAME='PUMA-SOLN.DAT', STATUS='NEW')
        open (9, NAME='input.dat', STATUS='old')

C Read input kinematic data

        read (9,*)
        read (9,*) fi0,th0,si0,px0,py0,pz0
        read (9,*) dt1,dd1,aa1,al1,bl1
        read (9,*) dt2,dd2,aa2,al2,bl2
```

```
      read (9,*) dt3,dd3,aa3,al3,bl3
      read (9,*) dt4,dd4,aa4,al4,bl4
      read (9,*) dt5,dd5,aa5,al5,bl5
      read (9,*)
      read (9,*) df6,th6,si6,px6,py6,pz6
      read (9,*)
      read (9,*) nobs,r,dangle,dlenth,magnx,magnl

      close (9)

C Adjust nominal values

      fi0=fi0+dangle
      th0=th0+dangle
      si0=si0+dangle
      px0=px0+dlenth
      py0=py0+dlenth
      pz0=pz0+dlenth

      dt2=dt2+dangle
      dt3=dt3+dangle
      dt4=dt4+dangle
      dt5=dt5+dangle

      al1=al1+dangle
      al2=al2+dangle
      al3=al3+dangle
      al4=al4+dangle
      al5=al5+dangle

      aa1=aa1+dlenth
      aa2=aa2+dlenth
      aa3=aa3+dlenth
      aa4=aa4+dlenth
      aa5=aa5+dlenth

      dd3=dd3+dlenth
      dd4=dd4+dlenth
      dd5=dd5+dlenth

      bl2=bl2+dangle

      df6=df6+dangle
      th6=th6+dangle
      si6=si6+dangle
      px6=px6+dlenth
      py6=py6+dlenth
      pz6=pz6+dlenth

C Limits on bar rotation

      phimax=0.0
      phimin=-180.0
      thetamax=180.0
      thetamin=-180.0

C Get random number seed

      write (6,*) 'Type in a 6-digit random number seed'
```

```
        read (5,*) iseed

C Write NOBS to PUMA-SOLN.DAT

        write (10,*) nobs

C Start of main loop

 1010   obs=obs+1

C Set joint angles to zero

        do i=1,n
        x(i)=0.0
        enddo

C Get random bar angles

 1000   call random (iseed,num)
        phi=phimin+(phimax-phimin)*num
        call random (iseed,num)
        theta=thetamin+(thetamax-thetamin)*num

C Calculate end point of the bar

        xb=r*cosd(theta)*cosd(phi)
        yb=r*sind(theta)*cosd(phi)
        zb=-r*sind(phi)

C Reacheability calculation

        if (z .lt. 0.0) go to 1000

        dx=xb-px0
        dy=yb-py0
        dz=zb-pz0
        rr=sqrt(dx*dx+dy*dy+dz*dz)
        if (rr .gt. (AA2+DD4+PZ6)) go to 1000

C Establish desired tool pose

        do ii=1,4
        do jj=1,4
                TDES(ii,jj)=0.0
        enddo
        enddo

        TDES(1,4)=xb
        TDES(2,4)=yb
        TDES(3,4)=zb
        TDES(4,4) = 1.0

C Call IMSL'ZXSSQ for inverse kinematic solution

        nsig=4
        eps=0.0
        delta=0.0
        maxfn=500
        iopt=1
        ixjac=ldfjac
        CALL ZXSSQ(puma_arm,m,n,nsig,eps,delta,maxfn,iopt,parm,x,
     &                 ssq,f,xjac,ixjac,xjtj,work,infer,ier)

C Check for singularities

        if (ssq .gt. 0.00001) goto 1000

C Check joint angles are within ranges

c       do j=1,6
c       write(6,*) j,x(j)
c       irev=int(x(j)/360.0)
c       x(j)=x(j)-irev*360.0
c       write(6,*)'ranged',irev,x(j)
c
c       enddo

C Print results to 2 decimal places

        write(6,*) obs,ssq
        WRITE (10,888) X(1), X(2), X(3), X(4), X(5), X(6)
 888    format ( 6f12.2 )

C Continue for other bar angles

        if (obs .lt. nobs) go to 1010

        CLOSE (10)

        END

C ************************************************************************

        SUBROUTINE PUMA_ARM (X,M,N,F)

C This subroutine calculates the non-linear function for the use of
C the IMSL routine ZXSSQ. It is the forward kinematic solution for
C the PUMA manipulator.

        INTEGER M, N
        REAL*8 X(N), F(M)

        INTEGER II, JJ
        REAL*8 DT1, DT2, DT3, DT4, DT5
        REAL*8 DD1, DD2, DD3, DD4, DD5
        REAL*8 AA1, AA2, AA3, AA4, AA5
        REAL*8 AL1, AL2, AL3, AL4, AL5
        REAL*8 BL1, BL2, BL3, BL4, BL5
        REAL*8 DF6, FI6, TH6, SI6, PX6, PY6, PZ6
        REAL*8 fi0, th0, si0, px0, py0, pz0

        REAL*8 TH1, TH2, TH3, TH4, TH5
        REAL*8 T0(4,4), T1(4,4), T2(4,4), T3(4,4), T4(4,4)
        REAL*8 T5(4,4), T6(4,4), trpy(4,4), txyz(4,4)
        REAL*8 TIMAT(4,4), T(4,4)
        REAL*8 disq,dis

        INTEGER I, J, K
        REAL*8 TDES(4,4), DANGLE, DLENTH, r
```

```
          COMMON /PDATA/ TDES, DANGLE, DLENTH, r
          COMMON /KIN/ DT1,DT2,DT3,DT4,DT5,
     &                 AL1,AL2,AL3,AL4,AL5,
     &                 AA1,AA2,AA3,AA4,AA5,
     &                 DD1,DD2,DD3,DD4,DD5,
     &                 BL1,BL2,BL3,BL4,BL5,
     &                 fi0,th0,si0,px0,py0,pz0,
     &                 DF6,TH6,SI6,PX6,PY6,PZ6

C Initialize the TIMAT matrix to an I matrix:

          DATA TIMAT/1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1/

C Initialize the T matrix to an I matrix

          DO II = 1,4
          DO JJ = 1,4
                  T(II,JJ) = TIMAT(II,JJ)
          ENDDO
          ENDDO

C Manipulator joint angles

          TH1 = DT1 + X(1)
          TH2 = DT2 + X(2)
          TH3 = DT3 + X(3)
          TH4 = DT4 + X(4)
          TH5 = DT5 + X(5)
          FI6 = DF6 + X(6)

C Compute the T matrices, T1 thru T6:

          call t3rpy (fi0,th0,si0, trpy)
          call t3xyz (px0,py0,pz0, txyz)
          call matmulc (t0,trpy,txyz)

          CALL TRANSFORM ( AL1, AA1, DD1, TH1, BL1, T1 )
          CALL TRANSFORM ( AL2, AA2, DD2, TH2, BL2, T2 )
          CALL TRANSFORM ( AL3, AA3, DD3, TH3, BL3, T3 )
          CALL TRANSFORM ( AL4, AA4, DD4, TH4, BL4, T4 )
          CALL TRANSFORM ( AL5, AA5, DD5, TH5, BL5, T5 )

          CALL t3rpy ( fi6, th6, si6, trpy )
          CALL T3XYZ ( PX6, PY6, PZ6, txyz )
          CALL matmulc ( t6, trpy, txyz )

C Compute the overall transformation, T:

          CALL MATMULA ( T, T0 )
          CALL MATMULA ( T, T1 )
          CALL MATMULA ( T, T2 )
          CALL MATMULA ( T, T3 )
          CALL MATMULA ( T, T4 )
          CALL MATMULA ( T, T5 )
          CALL MATMULA ( T, T6 )

C Calculate the function F

          f(1)=t(1,4)-tdes(1,4)
```

```
        f(2)=t(2,4)-tdes(2,4)
        f(3)=t(3,4)-tdes(3,4)

        RETURN
        END

C ************************************************************************

        SUBROUTINE RANDOM (x,z)

C This subroutine generates random numbers in the range 0-1
C using a supplied seed x, the returned random number being z.

        REAL FM, FX, Z
        INTEGER A, X, I, M
        DATA I/1/

        IF ( I .EQ. 0 ) GO TO 1000
        I=0
        M= 2 ** 20
        FM= M
        A= 2**10 + 3

 1000   X= MOD( A*X ,M)
        FX= X
        Z= FX/ FM

        RETURN
        END

C ******************************************************************
```

APPENDIX E. PROGRAM ID6 FOR BALL BAR

```
C ************************************************************************

        PROGRAM ID6

C Robot Identification using the Non-linear Least Squares method.
C Simulation data is read for the PUMA manipulator from
C the data file PUMA-soln.DAT

C Change parameter LDFJAC to change the number of observations,
C set LDFJAC =  Number of observations

        INTEGER LDFJAC, MM, M, NN, N, NSIG, MAXFN, IOPT, IXJAC, INFER, IER
        PARAMETER (LDFJAC=100, MM=LDFJAC, NN=30)

        REAL*8 FJAC(LDFJAC,NN), XJTJ((NN+1)*NN/2)
        REAL*8 PARM(4), F(LDFJAC), WORK((5*NN)+(2*MM)+((NN+1)*NN/2))
        REAL*8 X(NN)
        EXTERNAL PUMA_ARM

        REAL*8 DANGLE, DLENTH, TQ, DQ, EPS, DELTA, SSQ
        REAL*8 SQERR1, SQERR2
        REAL*8 fi0, th0, si0, px0, py0, pz0
        REAL*8 DT1, DT2, DT3, DT4, DT5
        REAL*8 DD1, DD2, DD3, DD4, DD5
        REAL*8 AA1, AA2, AA3, AA4, AA5
        REAL*8 AL1, AL2, AL3, AL4, AL5
        REAL*8 BL1, BL2, BL3, BL4, BL5
        REAL*8 FI6, DF6, TH6, SI6, PX6, PY6, PZ6

        REAL*8 U(NN,NN), V(NN,NN), TOL, S(NN), FJTJ(NN,NN)
        INTEGER NB, RJTJ, IPATH

        INTEGER I, J, K, NOBS, MAXNOBS
        REAL*8 magnx,magnl
        PARAMETER (MAXNOBS=100)
        REAL*8 TET1(MAXNOBS), TET2(MAXNOBS), TET3(MAXNOBS)
        REAL*8 TET4(MAXNOBS), TET5(MAXNOBS), TET6(MAXNOBS)
        REAL*8 R
        COMMON /PDATA/ NOBS, TET1, TET2, TET3, TET4, TET5, TET6, R

        COMMON /KIN/ DT1,DT2,DT3,DT4,DT5,
     &          AL1,AL2,AL3,AL4,AL5,
     &          AA1,AA2,AA3,AA4,AA5,
     &          DD1,DD2,DD3,DD4,DD5,
     &          BL1,BL2,BL3,BL4,BL5,
     &          fi0,th0,si0,px0,py0,pz0,
     &          DF6,TH6,SI6,PX6,PY6,PZ6

C Open data files for inputs and results

        OPEN (8, NAME='RESULT.DAT', STATUS='NEW')
        OPEN (9, NAME='PUMA-SOLN.DAT', STATUS='OLD')
        OPEN (10,NAME='INPUT.DAT', STATUS='OLD')

c Read input parameters

        read (10,*)
        read (10,*) fi0,th0,si0,px0,py0,pz0
        read (10,*) dt1,dd1,aa1,al1,bl1
        read (10,*) dt2,dd2,aa2,al2,bl2
        read (10,*) dt3,dd3,aa3,al3,bl3
        read (10,*) dt4,dd4,aa4,al4,bl4
        read (10,*) dt5,dd5,aa5,al5,bl5
        read (10,*)
        read (10,*) df6,th6,si6,px6,py6,pz6
        read (10,*)
        read (10,*) nobs,r,dangle,dlenth,magnx,magnl

        CLOSE (10)

C Initialize data variables

        X(1)=fi0
        X(2)=th0
        X(3)=si0
        X(4)=px0
        x(5)=py0
        x(6)=pz0

        X(7)=AA1
        X(8)=AL1

        X(9)=DT2
        X(10)=AA2
        X(11)=AL2
        X(12)=BL2

        X(13)=DT3
        X(14)=DD3
        X(15)=AA3
        X(16)=AL3

        X(17)=DT4
        X(18)=DD4
        X(19)=AA4
        X(20)=AL4

        X(21)=DT5
        X(22)=DD5
        X(23)=AA5
        X(24)=AL5

        x(25)=df6
        x(26)=th6
        x(27)=si6
        x(28)=px6
        x(29)=py6
        x(30)=pz6

        R=R+MAGNX

C Read simulated joint data and tool pose

        READ (9,*) NOBS

        DO J = 1, NOBS
          READ (9,*) TET1(J), TET2(J), TET3(J), TET4(J), TET5(J), TET6(J)
        ENDDO
        CLOSE (9)
```

```
C Call IMSL routine for non-linear identification

        NSIG=4
        EPS=0.0
        DELTA=0.0
        MAXFN=1000
        IOPT=1
        IXJAC=LDFJAC
        M=NOBS

        CALL ZXSSQ(PUMA_ARM,M,NN,NSIG,EPS,DELTA,MAXFN,IOPT,
     &             PARM,X,SSQ,F,FJAC,IXJAC,XJTJ,WORK,INFER,IER)

C Calc the jacobian transpose product and it's rank; rank(jTj)

        NB = NN
        IPATH = 00
        TOL = 1.E-3

        CALL DMXTXF(M, NN, FJAC, LDFJAC, NB, FJTJ, NN)
        CALL DLSVRR(NN, NN, FJTJ, NN, IPATH, TOL, RJTJ, S, U, NN, V, NN)

        OPEN (15,NAME='JTJ.DAT', STATUS='NEW')
        kk = 0
        k = 0
C write jTj to a file in column order

        write(15,*)'mat size(square)','rank'
        write(15,*)nn,rjtj
        write(15,*)
        write(15,*)'jTj in column order'
        do k = 1,NN
           do kk = 1,NN
                write(15,*) fjtj(kk,k)
           enddo
        enddo
        CLOSE (15)

C Save results to data file

        WRITE (8,*)
        WRITE (8,*) 'fi0, th0, si0, px0, py0, pz0'
        WRITE (8,*) X(1), X(2), X(3), X(4), x(5), x(6)
        WRITE (8,*)
        WRITE (8,*) 'DT1, DD1, AA1, AL1, BL1'
        WRITE (8,*) 0.0, 0.0, X(7), X(8), 0.0
        WRITE (8,*)
        WRITE (8,*) 'DT2, DD2, AA2, AL2, BL2'
        WRITE (8,*) X(9), 0.0, X(10), X(11), X(12)
        WRITE (8,*)
        WRITE (8,*) 'DT3, DD3, AA3, AL3, BL3'
        WRITE (8,*) X(13), X(14), X(15), X(16), 0.0
        WRITE (8,*)
        WRITE (8,*) 'DT4, DD4, AA4, AL4, BL4'
        WRITE (8,*) X(17), X(18), X(19), X(20), 0.0
        WRITE (8,*)
        WRITE (8,*) 'DT5, DD5, AA5, AL5, BL5'
        WRITE (8,*) X(21), X(22), X(23), X(24), 0.0
        WRITE (8,*)
        WRITE (8,*) 'DF6, TH6, SI6, PX6, PY6, PZ6'
        WRITE (8,*) x(25), x(26), x(27), x(28), x(29),  x(30)
 888    FORMAT ( 5F12.5 )
 889    FORMAT ( 6F12.5 )

        WRITE(8,*) 'R=',R
c Restore initial values of input parameters

        open (10,name='input.dat',status='old')

        read (10,*)
        read (10,*) fi0,th0,si0,px0,py0,pz0
        read (10,*) dt1,dd1,aa1,al1,bl1
        read (10,*) dt2,dd2,aa2,al2,bl2
        read (10,*) dt3,dd3,aa3,al3,bl3
        read (10,*) dt4,dd4,aa4,al4,bl4
        read (10,*) dt5,dd5,aa5,al5,bl5
        read (10,*)
        read (10,*) df6,th6,si6,px6,py6,pz6
        read (10,*)
        read (10,*) nobs,r,dangle,dlenth,magnx,magnl

        CLOSE (10)

C Calculate root mean square error in identification

        TQ = DANGLE
        DQ = DLENTH

C Error in identification (angular parameters)

        SQERR1 =
     &   (DT2+TQ-X(6))**2
     &  +(DT3+TQ-X(10))**2 +(DT4+TQ-X(14))**2
     &  +(DT5+TQ-X(18))**2
     &  +(AL1+TQ-X(5))**2  +(AL2+TQ-X(8))**2
     &  +(AL3+TQ-X(13))**2 +(AL4+TQ-X(17))**2
     &  +(AL5+TQ-X(21))**2 +(BL2+TQ-X(9))**2
     &  +(DF6+TQ-X(22))**2
        SQERR1 = DSQRT( SQERR1/11 )

C Error in identification (length parameters)

        SQERR2 =
     &  (AA1+DQ-X(4))**2   +(AA2+DQ-X(7))**2
     &  +(AA3+DQ-X(12))**2 +(AA4+DQ-X(16))**2
     &  +(AA5+DQ-X(20))**2 +(DD1+DQ-X(3))**2
     &  +(DD3+DQ-X(11))**2 +(DD4+DQ-X(15))**2
     &  +(DD5+DQ-X(19))**2 +(PZ6+DQ-X(24))**2
     &  +(PX6+DQ-X(23))**2
     &  +(px0+dq-x(1))**2   +(py0+dq-x(2))**2
        SQERR2 = DSQRT( SQERR2/13 )

        WRITE (8,*)
        WRITE (8,*) 'RMS PARMS (LENGTH), RMS PARMS (ANGLE)'
        WRITE (8,*) SQERR2, SQERR1
        WRITE (6,*) 'RMS PARMS (LENGTH), RMS PARMS (ANGLE)'
        WRITE (6,*) SQERR2,SQERR1
        WRITE (8,*)
        WRITE (8,*) 'INFER, IER,NOBS,NSIG,rank'
        WRITE (8,*)  INFER, IER,NOBS,NSIG,rjtj
        WRITE (6,*) 'INFER, IER,NOBS,NSIG,rank'
        WRITE (6,*)  INFER, IER,NOBS,NSIG,rjtj
        WRITE (8,*)

        CLOSE (8)

        END

C *********************************************************************

        SUBROUTINE PUMA_ARM (X, M, N, F)

C This subroutine calculates the non-linear function for the use of
C the IMSL routine ZXSSQ. It is the forward kinematic solution for
C the PUMA manipulator.

        INTEGER M, N
        REAL*8 X(N), F(M)

        INTEGER II, JJ

        REAL*8 fi0, th0, si0, px0, py0, pz0
        REAL*8 DT1, DT2, DT3, DT4, DT5
        REAL*8 DD1, DD2, DD3, DD4, DD5
        REAL*8 AA1, AA2, AA3, AA4, AA5
        REAL*8 AL1, AL2, AL3, AL4, AL5
        REAL*8 BL1, BL2, BL3, BL4, BL5
        REAL*8 fi6, df6, th6, si6, PX6, PY6, PZ6

        REAL*8 TH1, TH2, TH3, TH4, TH5
        REAL*8 T0(4,4), T1(4,4), T2(4,4), T3(4,4), T4(4,4)
        REAL*8 T5(4,4), T6(4,4), trpy(4,4), txyz(4,4)
        REAL*8 TIMAT(4,4), T(4,4)

        INTEGER I, J, K, NOBS, MAXNOBS
        PARAMETER (MAXNOBS=100)
        REAL*8 TET1(MAXNOBS), TET2(MAXNOBS), TET3(MAXNOBS)
        REAL*8 TET4(MAXNOBS), TET5(MAXNOBS), TET6(MAXNOBS)
        REAL*8 R, RR
        COMMON /PDATA/ NOBS, TET1, TET2, TET3, TET4, TET5, TET6, R

        COMMON /KIN/ DT1,DT2,DT3,DT4,DT5,
     &          AL1,AL2,AL3,AL4,AL5,
     &          AA1,AA2,AA3,AA4,AA5,
     &          DD1,DD2,DD3,DD4,DD5,
     &          BL1,BL2,BL3,BL4,BL5,
     &          fi0,th0,si0,px0,py0,pz0,
     &          DF6,TH6,SI6,PX6,PY6,PZ6

C Initialize the TIMAT matrix to an I matrix:

        DATA TIMAT/1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1/

C Set parameters for the manipulator:

      fi0 = X(1)
      th0 = X(2)
```

```
        si0 = X(3)
        px0 = X(4)
        py0 = x(5)
        pz0 = x(6)

        DT1 = 0.0
        DD1 = 0.0
        AA1 = X(7)
        AL1 = X(8)
        BL1 = 0.0

        DT2 = X(9)
        DD2 = 0.0
        AA2 = X(10)
        AL2 = X(11)
        BL2 = X(12)

        DT3 = X(13)
        DD3 = X(14)
        AA3 = X(15)
        AL3 = X(16)
        BL3 = 0.0

        DT4 = X(17)
        DD4 = X(18)
        AA4 = X(19)
        AL4 = X(20)
        BL4 = 0.0

        DT5 = X(21)
        DD5 = X(22)
        AA5 = X(23)
        AL5 = X(24)
        BL5 = 0.0

        DF6 = x(25)
        TH6 = x(26)
        SI6 = x(27)
        px6 = x(28)
        py6 = x(29)
        pz6 = x(30)

C Loop NOBS times

        K = 0
        DO J = 1, NOBS

C Initialize the T matrix to an I matrix

        DO II = 1,4
        DO JJ = 1,4
                T(II,JJ) = TIMAT(II,JJ)
        ENDDO
        ENDDO

C Manipulator joint angles

        TH1 = DT1 + TET1(J)
        TH2 = DT2 + TET2(J)
        TH3 = DT3 + TET3(J)
        TH4 = DT4 + TET4(J)
        TH5 = DT5 + TET5(J)
        FI6 = DF6 + TET6(j)

C Compute the T matrices, T1 thru T6:

        call t3rpy(fi0,th0,si0, trpy)
        call t3xyz(px0,py0,pz0, txyz)
        call matmulc(t0,trpy,txyz)

        CALL TRANSFORM ( AL1, AA1, DD1, TH1, BL1, T1 )
        CALL TRANSFORM ( AL2, AA2, DD2, TH2, BL2, T2 )
        CALL TRANSFORM ( AL3, AA3, DD3, TH3, BL3, T3 )
        CALL TRANSFORM ( AL4, AA4, DD4, TH4, BL4, T4 )
        CALL TRANSFORM ( AL5, AA5, DD5, TH5, BL5, T5 )

        CALL t3rpy ( fi6, th6, si6, trpy )
        CALL T3XYZ ( PX6, PY6, PZ6, txyz )
        CALL matmulc ( t6, trpy, txyz )

C Compute the overall transformation, T:

        CALL MATMULA ( T, T0 )
        CALL MATMULA ( T, T1 )
        CALL MATMULA ( T, T2 )
        CALL MATMULA ( T, T3 )
        CALL MATMULA ( T, T4 )
        CALL MATMULA ( T, T5 )
        CALL MATMULA ( T, T6 )

C Calculate the function F

        rr=dsqrt( t(1,4)*t(1,4)+t(2,4)*t(2,4)+t(3,4)*t(3,4) )
        f(j)=dabs( rr-r)

C End the do-loop for counter J

        ENDDO

C Compute RMS error

        sumsq=0.0
        do j=1, nobs
        sumsq=sumsq+f(j)*f(j)
        enddo
        rms=sqrt(sumsq/nobs)
        write (6,*) rms

        RETURN
        END

C ************************************************************************
```

LIST OF REFERENCES

1. Driels, M. R., "Using Passive End-Point Motion Constraints to Calibrate Robot Manipulators," *Journal of Dynamic Systems, Measurement, and Control,* vol. 115, p. 560, ASNE, 1993.

2. Driels, M. R., Swayze, W., Potter, S. A., "Full Pose Calibration of a Robot Manipulator Using a Coordinate Measuring Machine," *The International Journal of Advanced Manufacturing Technology,*vol. 8, p. 34, Springer-Verlag London Ltd., 1993.

3. Driels, M. R., Pathre, U. S., "Significance of Observation Strategy on the Design of Robot Calibration Experiments," Journal of Robotic Systems, vol. 7, no. 2, p. 198,John Wiley and Sons, Inc., 1990.

4. Driels, M. R., "Using Passive End-Point Motion Constraints to Calibrate Robot Manipulators," *Journal of Dynamic Systems, Measurement, and Control,* vol. 115, p. 561, ASNE, 1993.

5. Swayze, W. E., "Modelling Experimental Procedures for Manipulator Calibration," p. 2, Naval Postgraduate School, 1991.

6. Driels, M. R., "Using Passive End-Point Motion Constraints to Calibrate Robot Manipulators," *Journal of Dynamic Systems, Measurement, and Control,* vol. 115, pp. 560-565, ASNE, 1993.

7. Potter, S. A., "Full Pose and Partial Pose Calibration of a Six Degree of Freedom Robot Manipulator Arm," pp. 1-120, Naval Postgraduate School, 1991.

8. Swayze, W. E., "Modelling Experimental Procedures for Manipulator Calibration," pp. 1-198, Naval Postgraduate School, 1991.

9. Paul, R. P., *Robot Manipulators: Mathamatics, Programming and Control,* pp. 9-83, The MIT Press, 1982.

10. Mooring, B. W., Roth, Z. S., Driels, M. R., *Fundamentals of Manipulator Calibration,* John Wiley and Sons, Inc., 1991.

11. Paul, R. P., *Robot Manipulators: Mathamatics, Programming and Control,* p. 45, The MIT Press, 1982.

12. Mooring, B. W., Roth, Z. S., Driels, M. R., *Fundamentals of Manipulator Calibration,* p. 45, John Wiley and Sons, Inc., 1991.

13. Driels, M. R., Swayze, W., Potter, S. A., "Full Pose Calibration of a Robot Manipulator Using a Coordinate Measuring Machine," *The International Journal of Advanced Manufacturing Technology,* vol. 8, p. 36, Springer-Verlag London Ltd., 1993.

14. Driels, M. R., Pathre, U. S., "Significance of Observation Strategy on the Design of Robot Calibration Experiments," *Journal of Robotic Systems,* vol. 7, no. 2, p. 201,John Wiley and Sons, Inc., 1990.

15. Mooring, B. W., Roth, Z. S., Driels, M. R., *Fundamentals of Manipulator Calibration,* pp. 109-113, John Wiley and Sons, Inc., 1991.

16. Mooring, B. W., Roth, Z. S., Driels, M. R., *Fundamentals of Manipulator Calibration,* pp. 132-140, John Wiley and Sons, Inc., 1991.

17. Paul, R. P., *Robot Manipulators: Mathamatics, Programming and Control,* pp. 85-92, The MIT Press, 1982.

18. Mooring, B. W., Roth, Z. S., Driels, M. R., *Fundamentals of Manipulator Calibration,* pp. 41-44, John Wiley and Sons, Inc., 1991.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center..2
   Cameron Station
   Alexandria, Virginia 22304-6145

2. Library, Code 52 2
   Naval Postgraduate School
   Monterey, California 93943-5101

3. Naval Engineering Curricular Office, Code 34 1
   Department of Mechanical Engineering
   Naval Postgraduate School
   Monterey, California 93943-5000

4. Department Chairman, Code ME...1
   Department of Mechanical Engineering
   Naval Postgraduate School
   Monterey, California 93943-5000

5. Professor Morris Driels, Code MEDr 1
   Department of Mechanical Engineering
   Naval Postgraduate School
   Monterey, California 93943-5000

6. LT Robert Burger, USN 2
   9204 Waterford Drive
   College Station, Texas 77845